INTELLIGENT FUZZY CONTROLLER FOR SATELLITE GROUND STATION APPLICATIONS

Roy George, PhD

Department of Computer Science
Clark Atlanta University
Atlanta, GA 30314

September 1997

Final Report

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

19971210 097

DTIC QUALITY INSPECTED 2
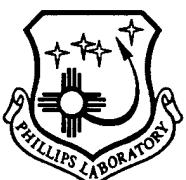
PHILLIPS LABORATORY
Space Technology Directorate
**AIR FORCE MATERIEL COMMAND**
KIRTLAND AIR FORCE BASE, NM 87117-5776

PL-TR-97-1119

ROSS WAINWRIGHT
Project Manager

FOR THE COMMANDER

STEPHEN STOVER, LT COL, USAF
Chief, Space System Technologies
Division

BRUCE A. THIEMAN, COL, USAF
Deputy Director, Space Technology

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1997	3. REPORT TYPE AND DATES COVERED Final; 9/95 to 9/97

4. TITLE AND SUBTITLE

Intelligent Fuzzy Controller for Satellite Ground Station Applications

5. FUNDING NUMBERS
C: F29601-95-K-0022
PE: 62601F
PR: 8809
TA: VT
WU: OH

6. AUTHOR(S)

Roy George, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Computer Science
Clark Atlanta University
Atlanta, GA 30314

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Phillips Laboratory
3550 Aberdeen Ave. SE
Kirtland AFB, NM 87117-5776

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

PL-TR-97-1119

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

The Department of Computer Science at Clark Atlanta University investigated the use of genetic algorithms as a technique for automating the development of fuzzy logic controllers. The derivation of fuzzy controller rule-bases is relatively straightforward; however, the tuning of these controllers is a difficult process. In the first phase of this research, a genetic algorithm was used to tune the fuzzy controller. The genetic algorithm searches through the space of all membership functions to select the functions that produce the best control action. In the second phase of this project, the genetic algorithm was used to automatically derive the rule-base and membership functions. A full-featured research prototype was developed. The methodology and prototype were validated on typical control and classification problems. This research establishes a methodology for the rapid development of robust knowledge-based control systems in complex, poorly understood domains.

14. SUBJECT TERMS
Knowledge-based control systems, fuzzy logic controller, fuzzy logic rulebase, genetic algorithm, real valued genetic algorithm, automated controller tuning

15. NUMBER OF PAGES
94

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

Table of Contents

Executive Summary

The objective of this research was to investigate a methodology for the automatic development of fuzzy controllers. Fuzzy controllers are a class of knowledge-based controllers that use fuzzy logic to model control actions. They are applicable in poorly understood or complex non-linear problems that defy control theory-based solutions. A number of space applications are amenable to solution using fuzzy controllers, including battery charger control, fault diagnosis, motion control, etc. The aim was to develop a generalized domain independent system.

The two principal components of a fuzzy controller are the fuzzy rule-base and the fuzzy membership functions. The primary objective was to develop an automatic tuning mechanism for the fuzzy rule-base (i.e., automatically tune the membership functions) using the genetic algorithm mechanism. A secondary objective was to investigate the applicability of genetic algorithms to automatically tune and generate both the fuzzy rule-base and membership functions. The genetic algorithm is a randomized search technique that uses an evaluation function to guide the search process.

As an initial study a fuzzy controller package, the TIL-Shell from Togai Infralogic, was studied. A typical control problem, the pendulum balancing problem, was implemented using the TIL-Shell. Issues relating to the application of genetic algorithms to fuzzy controllers were examined. A modified version of the genetic algorithm, the real valued genetic algorithm, was implemented.

In the first phase of this project, the rule-base (developed for the pendulum balancing problem) was tuned using the genetic algorithm. The aim was to tune the fuzzy rule-base by using the genetic algorithm to iteratively improve the membership functions. In other words, the genetic algorithm searches through the entire space of membership functions attempting to minimize the error in the output of the fuzzy controller. Excellent tuning results with high precision were obtained through this process. A drawback of this approach, however is that it presupposes the existence of a fuzzy rule-base.

In the second phase, the approach was generalized to automatically derive both the fuzzy rule-base and the membership functions using the genetic algorithm. The genetic algorithm searches through a space of all possible rules and membership functions to derive a fuzzy controller with ideal performance. As previously, the genetic search is guided by the minimization of the error between the expected output and the actual output produced by the fuzzy controller. The full-featured system prototype was developed and its performance verified with good results on both control and classification problems. Though scale-up issues were not extensively investigated in this effort, this approach promises good results for large, poorly understood, and complex control applications.

1. Fuzzy Logic Controllers

Fuzzy logic has been widely applied in industrial control applications. This has been the result of large complex system applications that make precise measurements and modeling difficult. Knowledge-based controllers using linguistic expressions and terms are therefore a viable alternative. A qualitative modeling technique that can been model the human problem solving process is fuzzy logic. The fundamental idea behind fuzzy logic controllers is the use of fuzzy logic to model and develop industrial control applications. Unlike traditional knowledge-based systems, fuzzy control does not use the iterative execution of instructions, but rather the execution of a number of rules in parallel – fuzzy inference.

The following represents a typical relationship (rule) expressed in a fuzzy system, a "ground" truth (condition), and the inference (action) that may be derived from these relationships.

Rule	*IF the distance between cars is small, reduce speed*
Condition	*The distance between cars is 100 ft*

Action	*Reduce Speed Considerably*

The action is the result of inference. In a fuzzy controller, there would be a number of applicable rules that require each inference rule to be combined in some manner and the result converted to a numerical value. This numerical value is then used to control the system.

There are four classical methods of designing a fuzzy controller [Terano]:

- Extraction of Domain Information from Experts: The experience of skilled operators or the knowledge of control engineers is assimilated and brought together in the form of IF-THEN rules.

- Building Operator Models: In cases where the skill of operators is hard to quantify, their actions are modeled using IF-THEN rules.

- Deriving Rules Through Learning: If a plant model exists, fuzzy rules can be learned from an environment in which there are no experts.

- Development of a Fuzzy Model of the Plant: A fuzzy model of the system is built and fuzzy control rules are derived from the control objectives and the plant model.

1.1 Fuzzy Controller Development

The principal components of the fuzzy controller are the fuzzy rule base, the inference mechanism, and the input and output interfaces. The rule base defines the actions of the fuzzy controller. There are four steps in constructing a fuzzy rule base:

- Identify and name the input variables and their ranges.

- Identify and name the output variables and their ranges.

- Define a fuzzy membership function for each of the input and output variables. The membership functions assign a value of inclusion to every possible input value. The inclusion is with respect to predefined linguistic ranges that cover the entire or parts of the input and output ranges.

- Construct a rule base that will govern the controllers operation.

- Determine how the control actions will be combined to form the executed action (defuzzification).

A number of alternate defuzzification techniques have been proposed. In this project, we utilized a Mamdani-Type Fuzzy Controller [Mamdani] that uses centroid defuzzification. The centroid defuzzification rule is given below:

$$Output = \Sigma \; x_i * \mu_A[x_i] / \Sigma \; \mu_A[x_i]$$

where x_i, is the signal and $\mu_A[x_i]$, is the membership of x_i, in the linguistic quantifier,

A.

Other commonly used defuzzification schemes include the Takagi-Sugano model [Takagi]. The difference between the Sugeno and the Mamdani model is that the Sugeno model has a mathematical function as the rule consequent, whereas in the Mamdani model each rule has a fuzzy consequent.

The principal components of the fuzzy controller are shown in Figure 1.
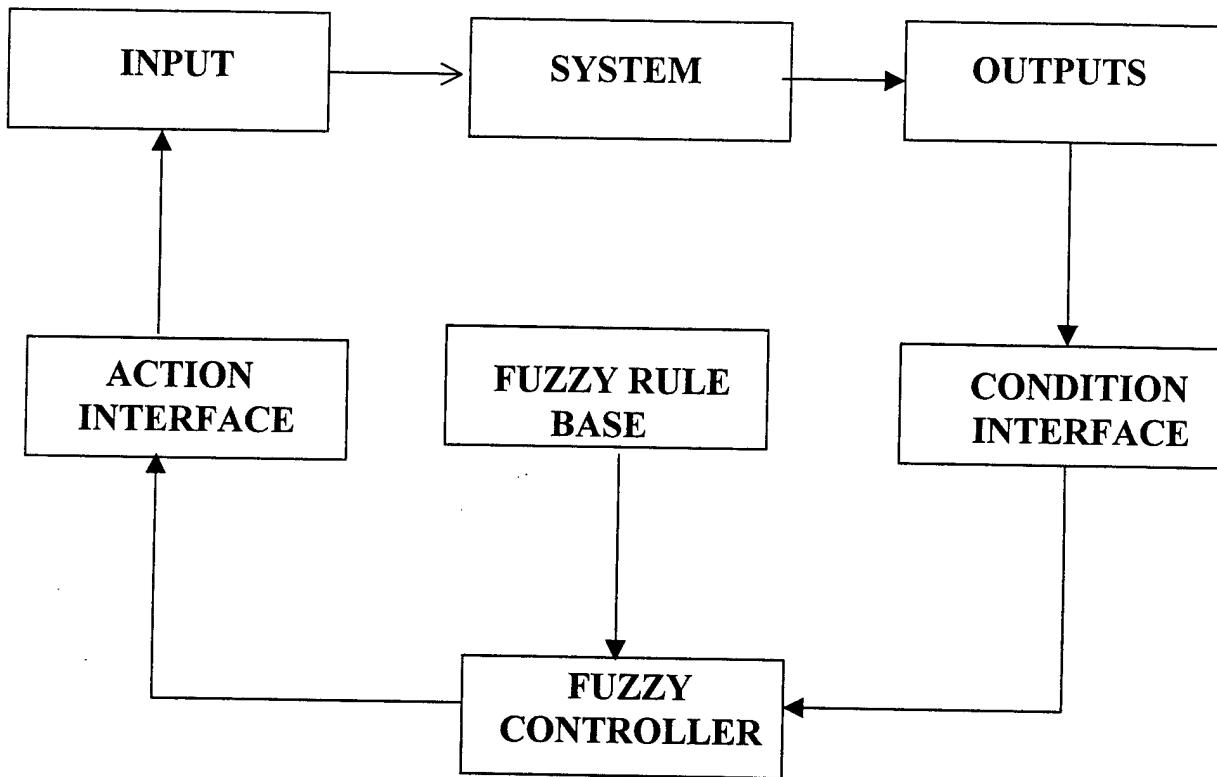
Figure 1: Block Diagram of a Fuzzy Controller

The centroid defuzzification algorithm is implemented as below. Note that a fuzzy controller simulation has to be implemented as part of the genetic algorithm evaluation function.

```c
void calcMemAreaCent(member, lower, upper, Area, Centroid)
struct memberObject *member;
double lower;
double upper;
double *Area;
double *Centroid;
{
        double AreaOne;
        double AreaTwo;
        double CentroidOne;
        double CentroidTwo;

        switch(member → type)
        {
          case TRIANGULAR:
                *Area = 0.5 (member→rightVertex - member→leftVertex)(member→mu);
                CentroidTwo = ((member→rightVertex - member→leftVertex) +
                        (member→rightVertex - member→leftVertex);
                *Centroid = member→rightVertex – CentroidTwo;

          case S_FUNCTION:
                AreaOne = 0.5 (member→rightVertex - member→leftVertex)(member→mu);
                AreaTwo = (upper – member→centerVertex)*(member→mu);
                *Area = AreaOne + AreaTwo;
                CentroidOne = member→centerVertex – ((member→centerVertex –
                        member→leftVertex)/3.0);
                CentroidTwo = upper – (upper – member→centerVertex)/2.0;
                If (*Area > 0.0)
                {
                        Centroid = ((AreaOne*CentroidOne)+(AreaTwo*CentroidTwo))/(*Area);
                } else
                {
                        *Centroid = 0.0;
                }

          case Z_FUNCTION:
                AreaOne =  (member→centerVertex – lower) *(member→mu);
                AreaTwo = 0.5*(member→rightVertex – member→centerVertex)*(member→mu);
                *Area = AreaOne + AreaTwo;
                CentroidTwo = member→centerVertex – ((member→rightVertex –
                        member→centerVertex)/3.0);
                CentroidOne = lower + (member→centerVertex - lower)/2.0;
                If (*Area > 0.0)
                {
                        Centroid = ((AreaOne*CentroidOne)+(AreaTwo*CentroidTwo))/(*Area);
                } else
                {
                        *Centroid = 0.0;
                }

          default:
                fprintf(stderr, "\ncalcMemAreaCent: Error in the case. \n");
                exit(1);
                }
}
```

```
double CalcOutCentroid(Var)
/* this function calculated the overall centroid of a variable */
{
        int i;
        double Area, Centroid, TotalArea, TotalCentArea;

        Area = 0.0;
        Centroid = 0.0;
        TotalArea = 0.0;
        TotalCentArea = 0.0;

        For (i=0; i,Var→numOfMembers; i++)
        {
            calcMemAreaCent(&(Var→member[i], Var→lowerBound, Var→upperBound, &Area,
            &Centroid);
            TotalArea = TotalArea + Area;
            TotalArea = TotalCentArea + (Area*Centroid);
        }

        if (TotalArea > 0.0)
        {
            return(TotalCentArea/TotalArea);
        }
        else
        {
            return(0.0);
        }
}
```

Listing 1 : Genetic Algorithm Evaluation Function

1.2 Developing Fuzzy Controllers with the TIL-Shell

The TIL-Shell is a software development environment that provides a way of describing fuzzy logic systems, testing such systems through simulation, and compiling the system for further implementation on the target processor. A fuzzy system is defined in the TIL-Shell using a number of editors. Variables and rulebases are added and connected in the project editor. Membership functions are defined in the variable or member editors, and the rulebase is defined in the rulebase editor. The graphical project editor is the top level editor for creating and manipulating objects. It provides a block diagram view of the fuzzy system.

1.2.1 Pendulum Controller

The pendulum controller is a simplified control system for balancing a pendulum. The system is developed using the TIL-Shell. Appendix A, shows the rule base developed and the membership functions defined.

A total of nine rules were required for the pendulum controller. The rules are as follows:

IF error IS negative AND derror IS negative THEN current IS positive END
IF error IS negative AND derror IS zero THEN current IS positive END
IF error IS negative AND derror IS positive THEN current IS zero END

IF error IS zero AND derror IS negative THEN current IS positive END
IF error IS zero AND derror IS zero THEN current IS zero END
IF error IS zero AND derror IS positive THEN current IS negative END

IF error IS positive AND derror IS negative THEN current IS zero END
IF error IS positive AND derror IS zero THEN current IS negative END
IF error IS positive AND derror IS positive THEN current IS negative END

The definitions of the fuzzy functions (in this case *positive, zero,* and *negative*, for each of the three variables, *error, derror,* and *current*) directly affect the performance of the controller. The tuning employed here consists of manually changing the definition of each fuzzy function and quantifying the effect on the controlled variable. This process is laborious and infeasible for large controllers.

2. Genetic Algorithms

2.1 Background

Genetic Algorithms (GAs) are a class of iterative, randomized search procedures capable of adaptive and robust search over a wide range of space topologies. They are modeled after the adaptive emergence of biological species from evolutionary mechanisms [Holland]. GAs have been successfully applied in such diverse fields as image processing, scheduling, and engineering design. Detailed introduction to GAs and their applications can be found in [Goldberg, Davis, Eberhart].

The terminology used in GAs is mostly borrowed from the field of genetics. A single solution is called an **organism** and the set of solutions is termed a **population**. A **generation** denotes a population during an iteration. A string representation of an organism is called a **chromosome**. Each substring representing a feature of the solution is called a **gene**. The quality of an organism is computed through an **evaluation** or **fitness function**. Two mechanisms of reproduction are used- 1. **crossover**, and 2. **mutation**. In crossover, two parent organisms are spliced together to create a new

organism. In mutation, arbitrary bits in the organism are flipped. Mutation is generally performed at a much lower rate than crossover.

The operational characteristic of GAs can be represented by the following algorithm:

- *Create a random population of initial solutions*
- *Compute the fitness of each organism*
- *Until Convergence Do:*
  - *Create next generation by stochastically applying*
    - *Fitness Propotional Selection*
    - *Crossover*
    - *Mutation*
  - *On members of the current population*
    - *Compute fitness of members of the new population*
    - *Replace current population by the new population*
- *End Do*

2.2 Evaluation of GA Packages

Two public domain GA packages, SGA-C and Splicer, were evaluated for robustness, ease of usage, and platform dependencies. Splicer has a modular architecture that includes a Genetic Algorithm kernal, Representation Modules, and a User Interface Libarary. It requires a UNIX platform with X-Windows. SGA-C is a text-based extension of the Simple GA program [Goldberg]. The primary platform is UNIX. Comparative evaluation of the two packages resulted in the selection of SGA-C. Splicer had a number of run-time problems associated with the graphical user interface. SGA-C had better performance on sample problems and displayed greater robustness.

Both SGA-C and Splicer are based on binary representation of the problem domain. The mutation operator has significant effect on the solution. The objective of mutation is to achieve a local perturbation that moves the solution out from a local minima. The binary

9

nature of this process creates large scale effects on the solution rather than the effect of noise. This effect is known as the binary hill problem. A solution to this is to use a real valued GA (REGA) where mutation is achieved by the addition of noise to the solution. Several previous studies have noted the sensitivity of membership functions and shapes on the behavior of the fuzzy controller. The use of a real valued GA could reduce the sensitivity of the tuning process. A real valued GA was developed for the UNIX platform (Appendix B).

3. Tuning Fuzzy Logic Controllers Using Genetic Algorithms

A number of studies have shown that Fuzzy Logic Controllers (FLC) are sensitive to the characteristics of the membership function. Determining the membership function is an iterative process and requires engineering compromises. Automatic derivation of membership functions can ease this situation considerably. In this phase of the project, we used the genetic algorithm paradigm to automatically derive the membership functions. Before the mechanics of this approach are discussed, the following points have to be noted:

- Fuzzy sets of a input or output variable are ordered into regions of differing areas. Ideally, the area covered by the fuzzy sets decreases as the domain converges on the desired operating region. This has the effect of reducing the number of rules necessary to control the system since a single rule may handle all the device states in the outlier regions.

- The degree of overlap at the region of preferred performance provides fine tuning and control by the rules. This ensures that multiple rules will execute as the problem state

moves tò the left or right of the operating region. For example, in the pendulum balancing problem, the high degree of overlap ensures that when the system is in the optimal state (the pendulum is balanced), any small changes are immediately detected and handled.

- The number of fuzzy functions is always an odd number.

The fundamental ideal behind using GAs to tune the FLC rule base is simple. Recall that the GA attempts to converge to near optimal solutions using the fitness function. In this application, the fitness function is the FLC. The GA searches the space of membership functions, selecting only those membership functions that provide improved control. The desired input-output value pairs are provided to the GA. The root mean square error between the output value generated by the simulated FLC (the evaluation function) and the actual output value is used to drive the genetic algorithm into tuning the membership functions further. In other words, we are attempting to tune a black box (using a GA) so that it exhibits the desired characteristics indicated by the input-output pairs. For best results the input-output pair data should be representative of the control surface.

Three classes of membership functions are considered by the genetic algorithms: the Z, triangular, and S functions. Any function may be specified by three vertices: a left anchor, a mid-point, and a right anchor. The GA searches domain of membership functions (vertices and shape) for those that give the lowest root mean square error (indicating closeness to the supplied input-output pairs). The rule base is the set of rules

derived in Section 1.2.1. Figure 2 represents the arrangement of vertices in a chromosome.

LA	M	RA							

◄ NEGATIVE ►◄— ZERO —►

◄——— ERROR ———►

LA : LEFT ANCHOR
M : MID-POINT
RA : RIGHT ANCHOR
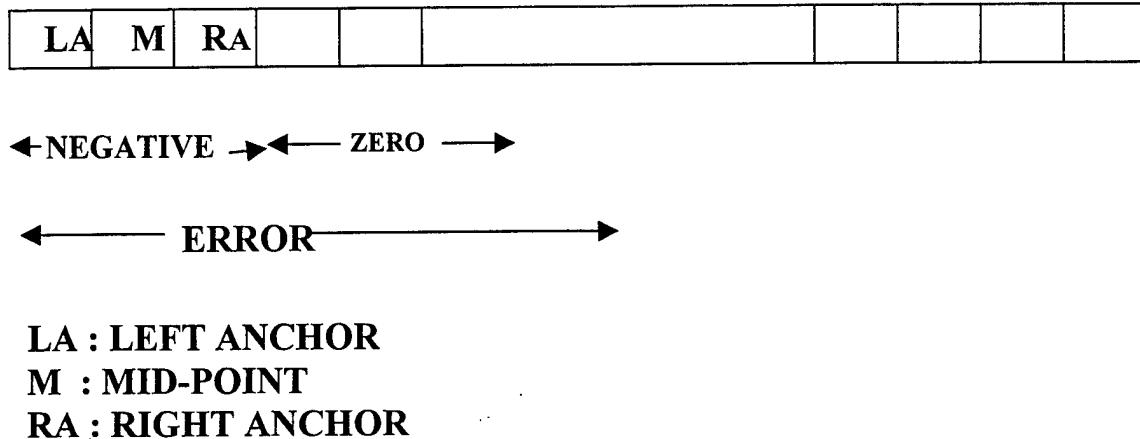
Figure 2 : Interpreting the Chromosome – Fuzzy Membership Functions

3.1 Results

A fuzzy controller tuner was developed using the SGA-C program. The pendulum-balancing problem was used to test the validity of the controller. Data was gathered by first defining the problem on the TIL-Shell and recording the input-output data pairs to the pendulum. Subsequently, this data was used as the input to the GA-based tuner. There are three components to the GA-based tuning process:

1. A control rulebase consisting of **IF X (AND/OR) Y THEN Z.**

2. A data file consisting of input-output data pairs.

3. A GA-based tuning mechanism.

The control rulebase and the fuzzy membership functions together form the evaluation function for the GA. The control rule base is the set of rules derived in 1.2.1. Appendix C gives a listing of the evaluation function (simulated GA). Appendix D shows sample runs of the tuner. For practical purposes a full run of the GA has not been included, but the sample indicates the continuous improvement of the control function (through the modification of the parameters of the membership functions). For instance, in Generation 4, the fitness is 0.430556, which by Generation 199 has improved to 0.992738. The maximum fitness possible is 1.0.

In comparison with the manual system, the automated system has two advantages. The obvious one is the ease of derivation of membership functions. The second is of precision. It is possible for the user to specify the precision (number of bits) required to describe each input or output membership function vertex. This gives higher precision than the TIL-Shell, where compiler precision is a limiting factor. The system that has been developed is very general. Any fuzzy controller (irrespective of the domain) may be tuned. The only requirements of the system are a representative sample of the input and output data pairs for the controller, and a set of fuzzy rules describing the control strategy. This approach would be most useful in situations where the domain expert that has already generated the control rulebase. The evolutionary approach eases the difficulty of tuning the existing rulebase.

4. Generating Fuzzy Controllers Using Genetic Algorithms

In the previous sections, we described the development of a fuzzy controller tuner using evolutionary mechanisms. The features of this tuner were the following:

1. Evolutionary learning of fuzzy membership functions for existing fuzzy controller

2. Tightly coupled GA-FCS mechanism

3. Study of alternate GA mechanisms to improve the learning rate.

The evolutionary tuner was tested on the pendulum balancing problem and highly accurate tuning was achieved. However, this technique is limited to problems for which a fuzzy rulebase already exists. It limits the scope of applicability considerably since this (approach) presupposes the existence of such a rulebase. There are many problems in classification, control, and decision making for which the structure of the problem is unknown or poorly understood (i.e., the fuzzy rules that govern the control/decision process are unknown due to the absence of domain expertise). This makes the evolutionary tuning process developed in the first phase of the project impracticable. The requirement here is for a system that can learn the structure of the problem (i.e., the fuzzy rules) and achieve the tuning of these rules automatically (as in the first phase).

The objective is to develop an evolutionary system that could automatically learn the structure of problems of arbitrary complexity. The problem can be subdivided into two related sub-problems:

1. Automatic generation of fuzzy rules

2. Tuning of the generated rules

14

Note, that the prime objective is the generation of articulated control systems (as opposed to systems generated using neural networks). For the automated generation of fuzzy rules a brute force technique was employed. The genetic organism representation consists of every possible combination of input and output linguistic quantifiers (chromosomal representation, one bit for each rule). The assumption made was that there are three membership functions for every input and output variable. Thus for a system consisting of m inputs and outputs, there are 4^m possible rules. The rule generation part of the organism will thus have 4^m bits.

Figure 3 is a representation of the GA organism. It is a generalization of Figure 2, with additional genetic material added for the representation of rules.

LA : LEFT ANCHOR
M : MID-POINT
RA : RIGHT ANCHOR
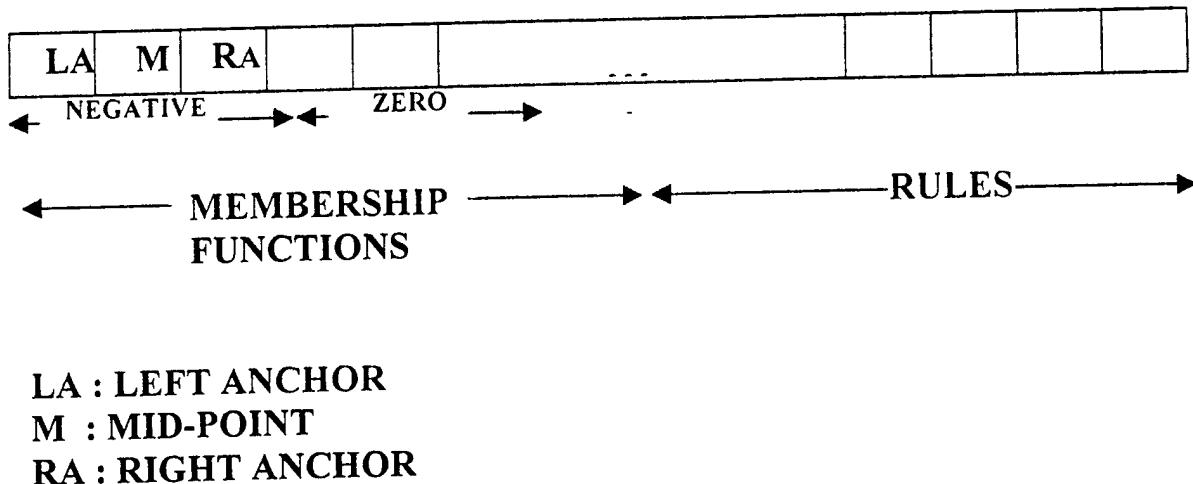
Figure 3: Interpreting the Chromosome – Fuzzy Rulebase and Membership Functions

The GA mechanism is used to search through the space of all possible rules and valid membership functions to obtain the best control system. An important objective is rule minimization. Solutions with minimal number of rules are preferred to controllers with a

larger number. A rule generator was designed and coded. The rules are indexed into a 2-dimensional linked list. The rules have been integrated into the genetic algorithm environment.

5. System Verification

The system was exhaustively tested on the pendulum balancing problem and a tank recognition problem. The pendulum balancing problem is a two input - one output problem. A complex non-linear relationship exists between the input and output variables.

5.1 Results

Pendulum Balancing Problem : The control problem is a two input (**Error** and **Derror**) - one output (**Current**) problem. The system generates the set of all possible rules that can describe the input-output relationship. This rule space (generated automatically by the system developed) is shown in Appendix E. There are three membership functions defined per variable - negative. zero. and positive. The rule space consists of 45 rules and 9 membership functions. The membership functions may be S-functions, Z-functions, or triangular.

The results of the evolutionary generation of the fuzzy control system are detailed in Appendix F. The system has reduced the rule set to six rules. The definition of the fuzzy memberships are also given in Appendix F. The performance results are very good with a Mean Square Error of 0.039.

Tank Recognition Problem: The problem considered is the recognition of tanks in a battlefield based on the audio sensor information. The data consists of three inputs and one output. A complex non-linear relationship exists between the input and output variables; however, it is not known what the relationship is. A set of apriori classified data is used to train and subsequently test the system. At the start, unlike the pendulum balancing problem (which was manually coded and tested), it was not known how many rules would be required to describe the system. The system generates the set of all possible rules that can describe the input-output relationship. This rule space (generated automatically by the system developed) is shown in Appendix F. Note that there are three membership functions defined per variable - negative, zero, and positive. The rule space consists of 189 rules and nine membership functions. The membership functions may be **S**-functions, **Z**-functions, or triangular. The results of the evolutionary fuzzy control system are detailed in Appendix F. Note that the system has reduced the rule set to seven rules. The performance results are reasonably good with a Mean Square Error of 0.414 and a Floor-Ceiling Mean Square Error of 0.498. A recognition rate of 76% is achieved on the test data. The larger error is due to the discretization of the output of the fuzzy controller required by the classification problem. Another issue is that the full tank recognition problem was not utilized (i.e., the entire input parameter set). This approach attempted the classification process using the three principal components of the problem. Further tuning and the use of the entire parameter set can improve the recognition rate.

A serious drawback with this approach is the combinatorial explosion (of possible rules) when the control problem has a large number of inputs and outputs. While the combinatorics is not a problem for the genetic algorithm, adequate representation of the rule set is problematic. In this implementation, rules are recursively generated for the complete set of input and output variables. Even small sets of these variables can result in the computational overhead of very large organisms.

6. Conclusions and Future Work

Theoretical issues regarding the application of genetic algorithms to the development of fuzzy logic controllers were studied in this research effort. This study has resulted in the development of a significant research prototype. A fuzzy rulebase was developed using a commercial fuzzy controller shell (the TIL-Shell) and tuned manually and automatically. The results of the automatic tuner are good, and deliver much higher precision than the manual tuner. There is also a significant reduction in the effort required in producing robust controllers. However, this approach presupposes the existence of a fuzzy controller rulebase. This is always not the case. To further generalize the approach, the prototype evolutionary fuzzy control system was enhanced (though not part of the initial project plan) to generate both fuzzy rules and tune the membership functions simultaneously.

In general terms, this approach is applicable across a wide spectrum of practical problems. The objective of this research was to investigate space applications for this technique. Two space applications suitable to the sponsor were considered. One of these

applications involved the control of smart structures. Data was not immediately available for this application. In the other application Lockheed Martin supplied controller simulation data for control of the null position of a solar drive encoder and to set the encoder initialization bits. The results of the evolutionary fuzzy controller were inconsistent with large mean square errors. Table 1 shows a portion of the Lockheed Martin data.

Time	+Y_Angle	+Y_Step	+Y_Plane_Error
0	0.1758	-1.0	-4.771
1	0.1758	0.0	-4.771
2	0.1758	1.0	-4.771
3	0.1758	2.0	-4.771
4	0.1758	3.0	-4.771

Table 1: Solar Drive Control Data

The +Y_Step is the output variable and the others the input variables. Two points about this data are noteworthy. First, Time is a monotonically increasing variable that determines +Y_Step. The evolutionary GA controller does not handle monotonically increasing or decreasing variables well. Second, the output is dependant on Time alone. The unsatisfactory results produced by the GA-based controller are related to the nature of this data. This application does not lend itself to a knowledge-based controller solution. However this is a feature of the specific problem domain rather than a drawback of the technique. Further interaction with the Phillips Laboratory would reveal a wider class of problems that can be easily solved by this technique.

Currently, the prototype is capable of generating valid fuzzy controllers in cases where the number of inputs and outputs is small. In general good results were obtained on both classification and control problems. A problem associated with the GA approach is that the space of valid controllers is very small, and typical GA operations of mutation and crossover can quickly move a controller from a valid to an invalid region. This does not give sufficient time for the GA to iteratively improve the solution. A solution could be the development of semantically driven mutation and crossover operations that narrow the search space rapidly.

As mentioned previously representation is a significant issue when dealing with large numbers of inputs and outputs. We have done preliminary investigation on an alternate methodology that overcomes the combinatorics in the representation – the Combs technique, with good results. In the Combs technique every input is mapped to every output. The resulting system has linear complexity. Tuning such a system is more difficult, but the GA-based tuner would ease this situation. This is a fruitful area for the investigation into complex fuzzy system applications.

References

[Davis]   Davis, L., (Ed.), **Handbook of Genetic Algorithms**, Van Nostrand Reinhold, New York, 1991.

[de Silva]   de Silva, C. W., "Considerations of Hierarchical Fuzzy Control," **Theoretical Aspects of Fuzzy Control**, Nguyen, H., Sugeno, M., Tong, R., and Yager, R. R. (Eds.), John Wiley & Sons, NY, (1995).

[Eberhart]   Eberhart, Russ, Simpson, Pat, and Dobbins, Roy, **Computational Intelligence PC Tools**, AP Professional, Cambridge, MA (1996).

[Goldberg]   Goldberg, D. E., **Genetic Algorithms in Search, Optimization, and Machine Learning**, Addison-Wesley, Reading, MA, 1989.

[Holland]   Holland, J. H., **Adaptation in Natural and Artificial Systems**, University of Michigan Press, Ann Arbor, MI, 1975.

[Mamdani]   Mamdani, E., and Assilian, S., "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," **International Journal of Man-Machine Studies**, 7(1), pp. 1- 13, 1975.

[Takagi]   Takagi, T, and Sugeno, M., "Fuzzy Identification of Systems and its Applications to Modeling and Control," **IEEE Trans. on Systems, Man, and Cybernetics**, 15(1), pp. 116-132, 1985.

[Terano]   Terano, Toshiro, Asai, Kyogi, and Sugeno, M.., **Applied Fuzzy Systems**, AP Professional, Cambridge, MA, 1994.

Appendix A

Pendulum Controller Developed Using the TIL-Shell

```
PROJECT Project1
  OPTIONS
    ICONCOLOR=12632256
    MODE="NORMAL"
    CHANGEID=2690521803
  END


/* The membership functions were defined using the membership editor */
/* Membership functions for Error */
  VAR Error
    OPTIONS
      ICONPOS=0.5,0.5
      FULLGRAPHICS="ON"
      GRIDSHOW="OFF"
      GRIDSNAP="OFF"
      GRIDSPACE=0.2,0.2
      NUMBER=3
      SCALE=50
      TOUCHED="ON"
    END
    TYPE float
    MIN -1
    MAX 1

    MEMBER N
      OPTIONS
        ICONCOLOR=16711680
      END
      POINTS -1,1 -0.58024691358,0.989583333333 0.037037037037,0
    END

    MEMBER Z
      OPTIONS
        ICONCOLOR=65407
      END
      POINTS -0.446913580247,0 0.140740740741,1 0.5,0
    END

    MEMBER P
      OPTIONS
        ICONCOLOR=255
```

22

```
      END
      POINTS 0.0172839506173,0 0.264197530864,1 1,1
    END
  END
/* Membership functions for dError */
  VAR dError
    OPTIONS
      ICONPOS=0.5,2.5
      GRIDSHOW="OFF"
      GRIDSNAP="OFF"
      GRIDSPACE=0.2,0.2
      NUMBER=3
      SCALE=50
      TOUCHED="ON"
    END
    TYPE float
    MIN -1
    MAX 1

    MEMBER N
      OPTIONS
        ICONCOLOR=16711680
      END
      POINTS -1,1 -0.257731958763,1 0.0103092783505,0
    END

    MEMBER Z
      OPTIONS
        ICONCOLOR=65407
      END
      POINTS -0.365979381443,0 -0.20618556701,0.987341772152 0.5,0
    END

    MEMBER P
      OPTIONS
        ICONCOLOR=255
      END
      POINTS -0.0257731958763,0 0.515463917526,1 1,1
    END
  END

/* Membership functions for Current */
  VAR Current
    OPTIONS
      ICONPOS=3,1.5
```

```
            GRIDSHOW="OFF"
            GRIDSNAP="OFF"
            GRIDSPACE=0.2,0.2
            NUMBER=3
            SCALE=50
            TOUCHED="ON"
        END
        TYPE float
        MIN -1
        MAX 1

        MEMBER N
            OPTIONS
                ICONCOLOR=16711680
            END
            POINTS -1,1 -0.551546391753,1 0.0257731958763,0
        END

        MEMBER Z
            OPTIONS
                ICONCOLOR=65407
            END
        POINTS -0.278350515464,0 -0.134020618557,0.987341772152 0.417525773196,0

        END

        MEMBER P
            OPTIONS
                ICONCOLOR=255
            END
            POINTS 0.0257731958763,0 0.5,1 1,1
        END
    END

    RULEBASE Pend_Rules
        OPTIONS
            ICONPOS=1.5,1.5
        END

        RULE Rule1
            IF (Error IS N) AND (dError IS N) THEN
                Current = P
        END

        RULE Rule2
```

```
        IF (Error IS Z) AND (dError IS N) THEN
          Current = P
      END

      RULE Rule3
        IF (Error IS P) AND (dError IS N) THEN
          Current = Z
      END

      RULE Rule4
        IF (Error IS N) AND (dError IS Z) THEN
          Current = P
      END

      RULE Rule5
        IF (Error IS Z) AND (dError IS Z) THEN
          Current = Z
      END

      RULE Rule6
        IF (Error IS P) AND (dError IS Z) THEN
          Current = N
      END

      RULE Rule7
        IF (Error IS P) AND (dError IS P) THEN
          Current = N
      END

      RULE Rule8
        IF (Error IS Z) AND (dError IS P) THEN
          Current = N
      END

      RULE Rule9
        IF (Error IS N) AND (dError IS P) THEN
          Current = Z
      END
    END

    DEBUG Debug1
    END

    CONNECT
      FROM Error
```

```
            TO Pend_Rules
        END

        CONNECT
            FROM dError
            TO Pend_Rules
        END

        CONNECT
            FROM Pend_Rules
            TO Current
        END
END
```

APPENDIX B
REAL VALUED GENETIC ALGORITH (ReGA)

```
/******************************************************************/
/* Real Valued Genetic Algorithm                               */
/* Written by : Radhakrishnan Srikanth, Roy George              */
/* Department of Computer Science                               */
/* Clark Atlanta University                                     */
/******************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#define eq ==
#define MAXINT 777777


typedef float GeneType;
typedef struct pop {  /* define item to hold the value and the fitness */
                GeneType *chromosome;
                float fitness;
                }PopType;

void Initialize();                  /* Initializes the members of the population */
void SrsSelect();                   /* Selects members of the Next Generation */
void CrossOver();           /* Selects and CrossesOver Chromosomes with a given probability */
void CrossOverChromosome(); /* Crossover generates two new children from given Parents */
void Mutate();                      /* Mutates Chromosomes in the population */
void MutateChromosome();        /* Mutates parts of the chromosome */
void CalculateFitness(); /* Calculates fitness of the chromsome */
void PrintReport();                 /* Prints Report */
void PrintChromosome();             /* Prints induvidual Chromosome */
void CorssOver();                   /* Performs CrossOver */
void CopyChromosome();                  /* Copies a give chromosome */
float ObjectiveFunction();          /* Calculates the fitness of a chromosome*/
void SRSelect();            /* Selects the new population */
void Statistics();                  /* Keeps the Statistics for each generation */
void InitializeData();              /* User written routine which reads data*/
void TestFunction();        /* Prints the output */


int PopulationSize;        /* Population Size */
int NumberOfParameters;             /* Number of Parameters per Chromosome */
```

```c
int seed;                                        /* The random seed that is used for generating
random numbers*/
float CrossOverProbability;      /* Stores the CrossOver Probility */
float MutateProbability; /* Stores the Mutation Probility */
float range=1;                                   /* Upper bound of range of the mutation amount */
int CrossOverStat=0;      /* Number of Crossovers */
int MutationStat=0;       /* Number of Mutations */
int GenerationNumber;            /* Keeps track of the generation */
FILE *graph;                             /* Keeps the statistics */
PopType BestEver;                /* Stores the Best ever Chromosome */
FILE *Data;                                      /* Data file for the neural network training
example */
int NumExamples;                 /* Number of examples in the file */
int NumInputs;           /* Number of input units in neurlal network*/
int NumOutputs;          /* Number of output units in neurlal network*/
int NumHidden;           /* Number of hidden units in neurlal network*/
float InputPattern[2][4];        /* Stores training pattern */
float Target[4];         /* Stores target pattern */




void main()
{

     int i;              /* Counters */
     PopType *Population1;    /* Contains current population of genes */
                 PopType *Population2;    /* Contains next population of genes */
     int MaxGenerations;              /* Maximum number of genertions */

     /* Prompt and Read number of parameters and the size of the population */

     printf("Enter the population size\n");
     scanf("%d",&PopulationSize);
     printf("Enter the number of parameters in each Chromosome\n");
     scanf("%d",&NumberOfParameters);

     /* allocate space for the members of the population */

     Population1 = (PopType *) malloc(PopulationSize*sizeof(PopType));
     Population2 = (PopType *) malloc(PopulationSize*sizeof(PopType));

     /* allocate space for each chromozome */

     for(i=0;i<PopulationSize;i++){
                 Population1[i].chromosome = (GeneType *) malloc(NumberOfParameters *
sizeof(GeneType));
                 Population2[i].chromosome = (GeneType *) malloc(NumberOfParameters *
sizeof(GeneType));
                 }
```

```c
        BestEver.chromosome = (GeneType *) malloc(NumberOfParameters * sizeof(GeneType));
                BestEver.fitness = (float)(-MAXINT);

                Initialize(Population1);
                printf("Enter Cross Over Probability (a number between 0-1)\n");
                scanf("%f",&CrossOverProbability);
                printf("Enter Mutation Probability (a number between 0-1)\n");
                scanf("%f",&MutateProbability);
                printf("Enter number of generations\n");
                scanf("%d",&MaxGenerations);

                /* Open the output file */

                if((graph = fopen("output","w+")) == NULL){
                    printf("Unable to open output file\n");
                    exit(0);
                }

    InitializeData(); /*Read Training Examples for the Neural Net */


    for(GenerationNumber=0;GenerationNumber<MaxGenerations;GenerationNumber++){
                CalculateFitness(Population1);
                SRSelect(Population1,Population2);
                CrossOver(Population2,Population1);
                Mutate(Population1);
                Statistics(Population1,&BestEver);
                }
                PrintReport(Population1);
                free(Population1);
                free(Population2);
                fclose(graph);

}


void Initialize(PopType *Population)
{
                int i,j;

                printf("Enter any integer for a seed\n");
                scanf("%d",&seed);
                srand(seed);

                for(i=0;i<PopulationSize;i++){
                    for(j=0;j<NumberOfParameters;j++)
                                Population[i].chromosome[j] = (GeneType)(((rand()%100)/3.3) -
15.12);
```

```c
                }
}


void CalculateFitness(PopType *Population)
{
                int i;

                for(i=0;i<PopulationSize;i++){
                    Population[i].fitness = ObjectiveFunction(Population[i].chromosome);
                }
}




void Statistics(PopType *Population, PopType *BestEver)
{
                int i;
                float Best;
                float Worst;
                float Average;

                Best = (float) (-MAXINT);
                Worst = (float) (MAXINT);
        Average = (float) 0;

                for(i=0;i<PopulationSize;i++){
                    Population[i].fitness = ObjectiveFunction(Population[i].chromosome);
/*                  if(Population[i].fitness > BestEver->fitness)
                            CopyChromosome(Population[i],BestEver); */
                    if(Population[i].fitness > Best)
                            Best = Population[i].fitness;
                    if(Population[i].fitness < Worst)
                            Worst = Population[i].fitness;
                    Average += Population[i].fitness;

                }
                Average = Average/PopulationSize;
                fprintf(graph,"%d %f %f\n",GenerationNumber,Best,Worst);
}


/* Prints report */

void PrintReport(PopType *Population)
{
                int i,j;

                for(i=0;i<PopulationSize;i++){
```

```c
                        printf("\nChromosome Fitness %f\n",Population[i].fitness);
                        for(j=0;j<NumberOfParameters;j++)
                                printf("Population[%2d].chromosome[%2d]                = 
%f\n",i,j,Population[i].chromosome[j]);
                        printf("\n");
                }
                printf("Total number of Crossovers %d\n",CrossOverStat);
                printf("Total number of Mutations %d\n",MutationStat);
                printf("Best Ever fitness is %f\n",BestEver.fitness);
                printf("Best Ever Chromosome is: \n");
/*              for(j=0;j<NumberOfParameters;j++)
                        printf("Population[%2d].chromosome[%2d]                =
%f\n",i,j,Population[i].chromosome[j]);
*/
                TestFunction(BestEver.chromosome);

}


void PrintChromosome(GeneType *chromosome)
{
     int j;
     for(j=0;j<NumberOfParameters;j++)
                printf("chromosome[%2d] = %f\n",j,chromosome[j]);
}


/* InitializeData function written by user to read appropriate Global Data */

void InitializeData()
{
     int i,j;

     if((Data = fopen("input.dat","r")) == NULL){
                printf("Error opening file input\n");
                exit(-1);
     }


     fscanf(Data,"%d %d %d",&NumInputs,&NumHidden,&NumOutputs);

     fscanf(Data,"%d",&NumExamples);

     for(i=0;i<NumExamples;i++){
                for(j=0;j<NumInputs;j++){
                    fscanf(Data,"%d",&InputPattern[j][i]);
                }
                fscanf(Data,"%d",&Target[i]);

     }
```

```c
        fclose(Data);

}


/* Objective function typically written by the user */

float ObjectiveFunction(GeneType *chromosome)
{

    int i,j,k,l;
    float net;
    float out;
    float hout[2];
    float fitness;    /* fitness of the chromosome */
    fitness = (float)0.0;

/*    for (i=0;i<NumberOfParameters;i++)
                fitness += chromosome[i];*/

    for(i=0;i<NumExamples;i++){
                k=0;
                for(l=0;l<NumHidden;l++){
                    net = (float)0.0;
                    for(j=0;j<NumInputs;j++){
                            net += InputPattern[i][j]*chromosome[k];
                            k++;
                    }
                    net = net-chromosome[k];
                    k++;
                    hout[l] = (float)net/(1+fabs(net));
                }
                for(l=0;l<NumHidden;l++){
                    out += hout[l]*chromosome[k];
                    k++;
                }
                out = out - chromosome[k];
                k++;
                out = (float)out/(1+fabs(out));
                if(fabs(out-Target[i]) < 0.5){
                    fitness += (float)1.0;
                }
    }
    return(fitness);


}
```

```c
void TestFunction(GeneType *chromosome)
{


    int i,j,k,l;
    float net;
    float out;
    float hout[2];

    for(i=0;i<NumExamples;i++){
            k=0;
            for(l=0;l<NumHidden;l++){
                net = (float)0.0;
                for(j=0;j<NumInputs;j++){
                        net += InputPattern[i][j]*chromosome[k];
                        k++;
                }
                net = net-chromosome[k];
                k++;
                hout[l] = (float)net/(1+fabs(net));
            }
            for(l=0;l<NumHidden;l++){
                out += hout[l]*chromosome[k];
                k++;
            }
            out = out - chromosome[k];
            k++;
            out = (float)out/(1+fabs(out));
            printf("output for pattern %d is %f\n",i,out);
    }


}

/* CrossOverChromosome, is a disruptive operator will take two
   induviduals in the population and produce two children */


void        CrossOverChromosome(GeneType        *parent1,GeneType        *parent2,GeneType
*child1,GeneType *child2)
{

    int CrossOverPoint;        /* Randomly Chosen Cross Over Point */
    int i;                     /* Counters */
```

```c
void TestFunction(GeneType *chromosome)
{


        int i,j,k,l;
        float net;
        float out;
        float hout[2];

        for(i=0;i<NumExamples;i++){
                k=0;
                for(l=0;l<NumHidden;l++){
                    net = (float)0.0;
                    for(j=0;j<NumInputs;j++){
                            net += InputPattern[i][j]*chromosome[k];
                            k++;
                    }
                    net = net-chromosome[k];
                    k++;
                    hout[l] = (float)net/(1+fabs(net));
                }
                for(l=0;l<NumHidden;l++){
                    out += hout[l]*chromosome[k];
                    k++;
                }
                out = out - chromosome[k];
                k++;
                out = (float)out/(1+fabs(out));
                printf("output for pattern %d is %f\n",i,out);
        }


}


/* CrossOverChromosome, is a disruptive operator will take two
   induviduals in the population and produce two children */


void     CrossOverChromosome(GeneType     *parent1,GeneType     *parent2,GeneType
*child1,GeneType *child2)
{

        int CrossOverPoint;          /* Randomly Chosen Cross Over Point */
        int i;                       /* Counters */
```

```c
/* pick a random point from 0 to Number of parameter */

srand(seed);

seed = rand();

CrossOverPoint = (int) (rand() % NumberOfParameters);
/* Cut and Splice */

for(i=0;i<CrossOverPoint;i++){
                child1[i] = parent1[i];
                child2[i] = parent2[i];
                }

for(i=CrossOverPoint;i<NumberOfParameters;i++){
                child1[i] = parent2[i];
                child2[i] = parent1[i];
                }
}


/* CopyChromosome copies a given chromosome to another */

void CopyChromosome(GeneType *OldCopy, GeneType *NewCopy)
{
    int i;

    for(i=0;i<NumberOfParameters;i++){
                NewCopy[i] = OldCopy[i];
    }

}


/* CrossOver will cross over different chromosomes selected with a
   probablity defined by the cross over probability */

void CrossOver(PopType *Selected, PopType *New)
{

    int i;
    float probability;
    int randomChromosome;

    srand(seed);
    seed = rand();
```

```
/* if the coin flip generates a number less than or equal to that of the
   cross over probability initiate crossover, else copy the induviduals
   into the new population */

        for(i=0;i<PopulationSize;i=i+2){
                   probability = ((float)(rand()%1000))/1000;
                   if(probability <= CrossOverProbability){
                        CrossOverStat++;

        CrossOverChromosome(Selected[i].chromosome,Selected[i+1].chromosome,New[i].chromoso
me,New[i+1].chromosome);
                        }
                   else{
        CopyChromosome(Selected[i].chromosome,New[i].chromosome);
        CopyChromosome(Selected[i+1].chromosome,New[i+1].chromosome);
                        }
        }/* end for */

        /* if the population size is not even pass a random chromosome to the
           new population */

        if (PopulationSize%2 != 0){
                   randomChromosome = rand()%PopulationSize;

        CopyChromosome(Selected[randomChromosome].chromosome,New[randomChromosome].c
hromosome);
        }

        CalculateFitness(New);

}


/* if the coin flip yeilds a probability less than or equal to the
   Mutation Probability mutate gene */

void MutateChromosome(GeneType *gene)
{
     int i;
     float probability;
     float mutateAmount;
     int power;

     srand(seed);
     seed = rand();
     for(i=0;i<NumberOfParameters;i++){
                   probability = ((float)(rand()%1000))/1000;
                   if(probability < MutateProbability){
                        MutationStat++;
```

```
                              /* Generates a random mutation amount in the user defined range
                                 and either adds or subtracts the generated noise to the original
                                 value */

                    mutateAmount = range*((float)(rand()%1000))/1000;
                    power = rand()%2;
                    mutateAmount = (float)pow((double)-1,(double)power)*mutateAmount;
                    gene[i] = gene[i]+(GeneType)(mutateAmount);

            }
        }/* end for */

}

/* Mutate Calls Mutate Chromosome */

void Mutate(PopType *Population)
{

    int i;

    for(i=0;i<PopulationSize;i++){
                    MutateChromosome(Population[i].chromosome);
    }

    CalculateFitness(Population);
}


/* SRSelect selets in one shot all the orgainsms for new population
   Assumes that CalculateFitness has already been called */

void SRSelect(PopType *Old, PopType *New)
{

    int i,j;
    float *slice;        /* keeps the % slice of the fitness for each organism */
    float TotalFitness=0; /* total fitness of the population */
    float WheelStop;     /* Keeps the random point where the Roulette Wheel stops */
    float delta;         /* amount by which the spoke in the wheel is displaced */
    int picked;          /* flag */
    FILE *fp;                    /* Temp file variable */


    if((fp = fopen("stat.dat","w+")) eq NULL){
        printf("Error: Cannot open file stat.dat\n");
        exit(0);
    }
```

37

```
                    /* Generates a random mutation amount in the user defined range
                       and either adds or subtracts the generated noise to the original
                       value */

                        mutateAmount = range*((float)(rand()%1000))/1000;
                        power = rand()%2;
                        mutateAmount = (float)pow((double)-1,(double)power)*mutateAmount;
                        gene[i] = gene[i]+(GeneType)(mutateAmount);


                    }
        }/* end for */

}


/* Mutate Calls Mutate Chromosome */

void Mutate(PopType *Population)
{

     int i;

     for(i=0;i<PopulationSize;i++){
                    MutateChromosome(Population[i].chromosome);
     }

     CalculateFitness(Population);
}



/* SRSelect selets in one shot all the orgainsms for new population
   Assumes that CalculateFitness has already been called */

void SRSelect(PopType *Old, PopType *New)
{

     int i,j;
     float *slice;         /* keeps the % slice of the fitness fo· each organism */
     float TotalFitness=0; /* total fitness of the population */
     float WheelStop,      /* Keeps the random point where the Roulette Wheel stops */
     float delta;          /* amount by which the spoke in the wheel is displaced */
     int picked;           /* flag */
     FILE *fp;                        /* Temp file variable */


     if((fp = fopen("stat.dat","w+")) eq NULL){
       printf("Error: Cannot open file stat.dat\n");
       exit(0);
     }
```

APPENDIX C
OBJECTIVE FUNCTION FOR AUTOMATICALLY TUNING THE
FUZZY CONTROLLER

```c
/*-----------------------------------------------------------*/
/*                                                           */
/* The routines below generate the control output  and compare */
/* the root mean square error of the output produced by the GA */
/* and the actual outputs.  This gives a measure of the quality */
/* of the fuzzy membership function generated by the GA      */
/* Sam Collins, Dept. of Computer Science, CAU               */
/*-----------------------------------------------------------*/



#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "external.h"


double evalTriMu(left, center, right , inputValue)
/* this function returns the truth value associated with the triangle */
double                  left;
double                  center;
double                  right;
double                  inputValue;
{
        double slope;
        double yInt;
        if((inputValue > left)&&(inputValue <= center))
        {
                slope = 1.0/(center - left);
                yInt = -slope * left;
        }else{
                slope = 1.0/(center-right);
                yInt = -slope * right;
        }
        return(slope*inputValue + yInt);

}

void evalInMemMu(member, inputValue)
/* This function is used to evaluate the truth value of a member. */
struct memberObject        *member;
double                     inputValue;
{
        switch(member->type)
        {
                case TRIANGULAR:
                        if(inputValue == member->centerVertex)
                        {
                                member->mu = 1.0;
                        } else if((inputValue > member->leftVertex)&&
                                (inputValue < member->rightVertex))
                        {
                                member->mu = evalTriMu(member->leftVertex,
```

```c
                                                        member->centerVertex,
                                                        member->rightVertex,
                                                        inputValue);
                        }else{
                                member->mu = 0.0;
                        }
                        break;
        case S_FUNCTION:
                        if(inputValue >= member->centerVertex)
                        {
                                member->mu = 1.0;
                        }else if((inputValue > member->leftVertex)&&
                                (inputValue < member->centerVertex)){
                                member->mu = evalTriMu(member->leftVertex,
                                                        member->centerVertex,
                                                        member->centerVertex,
                                                        inputValue);
                        }else{
                                member->mu = 0.0;
                        }
                        break;
        case Z_FUNCTION:
                        if(inputValue <= member->centerVertex)
                        {
                                member->mu = 1.0;
                        }else if((inputValue > member->centerVertex)&&
                                (inputValue < member->rightVertex)){
                                member->mu = evalTriMu(member->centerVertex,
                                                        member->centerVertex,
                                                        member->rightVertex,
                                                        inputValue);
                        }else{
                                member->mu = 0.0;
                        }
                        break;
        default:
                        fprintf(stderr,"\nevalInMemMu: Error in the case");
                        exit(1);
        }
}


void evalMuOfInVar(C, Sample)
/* this function is used to evaluate input variables mu */
struct controller *C;
int Sample;
{
        int i,j;
        for(i=0; i < C->numOfInputVar; i++)
        {
                for(j=0; j < C->inputVars[i].numOfMembers; j++)
                {
                        evalInMemMu(&(C->inputVars[i].member[j]),
                                        InputData[Sample][i]);
                }
        }
}



void ZeroMuOfOutVar(C)
/* this function is used to zero output variables mu */
```

```c
struct controller *C;
{
        int i,j;
        for(i=0; i < C->numOfOutputVar; i++)
        {
                for(j=0; j < C->outputVars[i].numOfMembers; j++)
                {
                        C->outputVars[i].member[j].mu=0.0;
                }
        }
}



double minDouble(value1, value2)
double value1;
double value2;
{
        if( value1 < value2)
                return(value1);
        else
                return(value2);
}

double maxDouble(value1, value2)
double value1;
double value2;
{
        if( value1 > value2)
                return(value1);
        else
                return(value2);
}


double applyRule(C,R, ruleP, ruleValue)
/* this function is used to apply a rule if it fires */
struct controller *C;
struct ruleObject *R;
struct ruleComp **ruleP;
double ruleValue;
{
        double truthValue;
        double truthForOr;
        truthValue = ruleValue;
        while((*ruleP)->type != THEN)
        {
        switch((*ruleP)->type)
        {
        case AND:
                truthValue = minDouble(truthValue,
                C->inputVars[(*ruleP)->varIndex].member[(*ruleP)->memIndex].mu);
                *ruleP = (*ruleP)->next;
                break;
        case OR:
                truthForOr =
                C->inputVars[(*ruleP)->varIndex].member[(*ruleP)->memIndex].mu;
                *ruleP = (*ruleP)->next;
        truthValue = maxDouble(truthValue, applyRule(C,R, ruleP, truthForOr));
                /* recursive function call should point to THEN when returns*/
                if((*ruleP)->type != THEN){
```

```c
                                fprintf(stderr, "\napplyRule: Error in recursive call! ");
                                exit(-1);
                        }
                        break;
                default:
                        fprintf(stderr,"\napplyRule: Error in case");
                        exit(1);
                }
        }
        return(truthValue);
}
void evalMuOfOutVar(C,R)
/* This function is utilizes the rules from the ruleBase to determine the */
/* mu of the output variables */
struct controller *C;
struct ruleObject *R;
{
        int             numOfRules;
        double          ruleValue;
        struct ruleComp *ruleP;
        for(numOfRules=0; numOfRules<R->numOfRules; ++numOfRules)
        {
                ruleP = &(R->rule[numOfRules]);
                ruleValue =
                    C->inputVars[ruleP->varIndex].member[ruleP->memIndex].mu;
                ruleP = ruleP->next;
                ruleValue = applyRule(C, R, &(ruleP), ruleValue);
                /* ruleP should be pointing to the 'THEN' component */
                if(ruleP->type != THEN)
                {
                    fprintf(stderr, "\nevalMuOfOutVar: Error should be THEN! ");
                    exit(-1);
                }
                C->outputVars[ruleP->varIndex].member[ruleP->memIndex].mu =
                    maxDouble(ruleValue,
                        C->outputVars[ruleP->varIndex].member[ruleP->memIndex].mu);
        }
}


void calcMemAreaCent(member, lower, upper, Area, Centroid)
struct memberObject *member;
double   lower;
double  upper;
double  *Area;
double  *Centroid;
{
        double AreaOne;
        double AreaTwo;
        double CentroidOne;
        double CentroidTwo;
        switch(member->type)
        {
          case TRIANGULAR:
                *Area = 0.5 * (member->rightVertex - member->leftVertex) *
                                  (member->mu);
                CentroidTwo =((member->rightVertex - member->leftVertex) +
                        (member->rightVertex - member->centerVertex))/3.0;
                *Centroid = member->rightVertex - CentroidTwo;
                break;
          case S_FUNCTION:
```

```c
                    AreaOne = 0.5 * (member->centerVertex - member->leftVertex) *
                                (member->mu);
                    AreaTwo = (upper - member->centerVertex) * (member->mu);
                    *Area = AreaOne + AreaTwo;
                    CentroidOne = member->centerVertex -
                                ((member->centerVertex - member->leftVertex)/3.0);
                    CentroidTwo = upper - (upper - member->centerVertex)/2.0;
                    if(*Area > 0.0)
                    {
                                *Centroid = ((AreaOne * CentroidOne) +
                                        (AreaTwo * CentroidTwo))/ (*Area);
                    }else{
                                *Centroid = 0.0;
                    }
                    break;
        case Z_FUNCTION:
                    AreaOne = ( member->centerVertex - lower) * (member->mu);
                    AreaTwo = 0.5 * (member->rightVertex - member->centerVertex) *
                                (member->mu);
                    *Area = AreaOne + AreaTwo;
                    CentroidTwo = member->centerVertex +
                                ((member->rightVertex - member->centerVertex)/3.0);
                    CentroidOne = lower + (member->centerVertex - lower)/2.0;
                    if(*Area > 0.0)
                    {
                                *Centroid = ((AreaOne * CentroidOne) +
                                        (AreaTwo * CentroidTwo))/ (*Area);
                    }else{
                                *Centroid = 0.0;
                    }
                    break;
        default:
                    fprintf(stderr,"\ncalcMemAreaCent: Error in the case.\n");
                    exit(1);
        }
}


double CalcOutCentroid(Var)
/* this function calculates the overall centroid of a variable */
struct varObject *Var;
{
        int i;
        double Area, Centroid, TotalArea, TotalCentArea;
        Area = 0.0;
        Centroid = 0.0;
        TotalArea = 0.0;
        TotalCentArea = 0.0;
        for(i=0; i < Var->numOfMembers; i++)
        {
                calcMemAreaCent(&(Var->member[i]), Var->lowerBound,
                        Var->upperBound, &Area, &Centroid);
                TotalArea = TotalArea + Area;
                TotalCentArea = TotalCentArea + (Area * Centroid);
        }

        if(TotalArea > 0.0)
        {
                return( TotalCentArea/TotalArea);
        }else{
                return(0.0);
```

```c
            } .
    }


    void getCalcOut(C, Samp)
    /* this function is used to evaluate the calculatef Out put for a sample */
    struct controller *C;
    int Samp;
    {
            int i;
            for(i=0; i < C->numOfOutputVar; i++)
            {
                CalculatedOutput[Samp][i] = CalcOutCentroid(&(C->outputVars[i]));
            }
    }


    double ZeroCalcOut(numOutVar)
    int numOutVar;
    {
            int numOfSamp,numOfOutputs;
            for(numOfSamp=0; numOfSamp<NumberOfSamples; ++numOfSamp)
            {
                    for(numOfOutputs=0; numOfOutputs<numOutVar; ++numOfOutputs)
                    {
                            CalculatedOutput[numOfSamp][numOfOutputs]=0.0;
                    }
            }
    }


    void calcOutputs(C,R)
    /* this function is used to evaluate the fitness of the controller */
    struct controller *C;
    struct ruleObject *R;
    {
            int Samp;
            ZeroCalcOut(C->numOfOutputVar);
            for(Samp=0; Samp < NumberOfSamples; Samp++)
            {
                    evalMuOfInVar(C, Samp);
                    ZeroMuOfOutVar(C);
                    evalMuOfOutVar(C,R);
                    getCalcOut(C, Samp);
            }
    }

    double ControllerFitness(numOutVar)
    /* This function is used to calculate the square of the differance of */
    /*   output data and the calculated output. Then compute the fitness.  */
    int numOutVar;
    {
            int numOfSamp,numOfOutputs;
            int i, j;
            double sum=0.0;
            double mse;
            for(numOfSamp=0; numOfSamp<NumberOfSamples; ++numOfSamp)
            {
                    for(numOfOutputs=0; numOfOutputs<numOutVar; ++numOfOutputs)
                    {
                            sum +=((CalculatedOutput[numOfSamp][numOfOutputs]-
                                    OutputData[numOfSamp][numOfOutputs])*
```

```
                          (CalculatedOutput[numOfSamp][numOfOutputs]-
                                  OutputData[numOfSamp][numOfOutputs]));
                  }
          }
                  mse = sqrt(sum/((double)NumberOfSamples)),
                  return(1.0/(1.0+mse));
}


void getData(numInVar, numOutVar)
/* This function is used to get the data from the DataFile.  The data will be */
/* used to calculate the fitness of the individual. */
int numInVar;
int numOutVar;
{
        FILE *fpGetData;
        int numOfSamp,numOfInputs,numOfOutputs;
        int i, j;
        if(numfiles == 0)
      fprintf(outfp," Enter the name of the datafile. ------------> ");
        fscanf(infp,"%s",DataFile);
        if((fpGetData=fopen(DataFile,"r")) == NULL)
        {
                fprintf(outfp,"Could not open %s\n",DataFile);
                exit(-1);
        }
        fscanf(fpGetData,"%d",&NumberOfSamples)
        /* allocate space for two dimensional array InputData */
        if((InputData = (double **)malloc
                (NumberOfSamples*sizeof(double *))) == NULL)
    {
                fprintf(stderr,"\nNot enough memory for InputData.\n");
                exit(-1);
    }
        for(i = 0; i < NumberOfSamples; i++)
        {
        if((InputData[i] = (double *)
                                malloc(numInVar*sizeof(double))) == NULL)
        {
                        fprintf(stderr,"\nNot enough memory for InputData.\n");
                        exit(-1);
                }
    }
        /* allocate space for two dimensional array OutputData */
        if((OutputData = (double **)
                malloc(NumberOfSamples*sizeof(double *))) == NULL)
    {
                fprintf(stderr,"\nNot enough memory for OutputData.\n");
                exit(-1);
        }
        for(j = 0; j < NumberOfSamples; j++)
        {
        if((OutputData[j] = (double *)
                                malloc(numOutVar*sizeof(double))) == NULL)
        {
                        fprintf(stderr,"\nNot enough memory for InputData.\n");
                        exit(-1);
                }
    }

/* allocate space for two dimensional array OutputData */
```

```c
                .         (CalculatedOutput[numOfSamp][numOfOutputs]-
                                OutputData[numOfSamp][numOfOutputs]));
                }
        }
                mse = sqrt(sum/((double)NumberOfSamples));
                return(1.0/(1.0+mse));
}


void getData(numInVar, numOutVar)
/* This function is used to get the data from the DataFile.  The data will be */
/* used to calculate the fitness of the indvidual. */
int numInVar;
int numOutVar;
{
        FILE *fpGetData;
        int numOfSamp,numOfInputs,numOfOutputs;
        int i, j;
        if(numfiles == 0)
        fprintf(outfp," Enter the name of the datafile. ----------> ");
        fscanf(infp,"%s",DataFile);
        if((fpGetData=fopen(DataFile,"r")) == NULL)
        {
                fprintf(outfp,"Could not open %s\n",DataFile);
                exit(-1);
        }
        fscanf(fpGetData,"%d",&NumberOfSamples)
        /* allocate space for two dimensional array InputData */
        if((InputData = (double **)malloc
                (NumberOfSamples*sizeof(double *))) == NULL)
        {
                fprintf(stderr,"\nNot enough memory for InputData.\n");
                exit(-1);
        }
        for(i = 0; i < NumberOfSamples; i++)
        {
        if((InputData[i] = (double *)
                                malloc(numInVar*sizeof(double))) == NULL)
        {
                        fprintf(stderr,"\nNot enough memory for InputData.\n");
                        exit(-1);
                }
        }
        /* allocate space for two dimensional array OutputData */
        if((OutputData = (double **)
                malloc(NumberOfSamples*sizeof(double *))) == NULL)
        {
                fprintf(stderr,"\nNot enough memory for OutputData.\n");
                exit(-1);
        }
        for(j = 0; j < NumberOfSamples; j++)
        {
        if((OutputData[j] = (double *)
                                malloc(numOutVar*sizeof(double))) == NULL)
        {
                        fprintf(stderr,"\nNot enough memory for InputData.\n");
                        exit(-1);
                }
        }

/* allocate space for two dimensional array OutputData */
```

```c
        if((CalculatedOutput = (double **)
                malloc(NumberOfSamples*sizeof(double *))) == NULL)
    {
                fprintf(stderr,"\nNot enough memory for CalculatedOutput.\n");
                exit(-1);
    }
        for(j = 0; j < NumberOfSamples; j++)
        {
        if((CalculatedOutput[j] = (double *)
                                malloc(numOutVar*sizeof(double))) == NULL)
            {
                fprintf(stderr,"\nNot enough memory for CalculatedOutput.\n");
                        exit(-1);
            }
    }
        for(numOfSamp=0; numOfSamp<NumberOfSamples; ++numOfSamp)
        {
                for(numOfInputs=0; numOfInputs<numInVar; ++numOfInputs)
            {
                    fscanf(fpGetData,"%lf",&InputData[numOfSamp][numOfInputs]);
            }
        for(numOfOutputs=0; numOfOutputs<numOutVar; ++numOfOutputs)
            {
                    fscanf(fpGetData,"%lf",&OutputData[numOfSamp][numOfOutputs]);
            }
        }

        fclose(fpGetData);/*Close the file */
}/* End of function getData. */

void printData(numInVar, numOutVar)
/* This function is used to print the data from the DataFile. */
int numInVar;
int numOutVar;
{
        int numOfSamp,numOfInputs,numOfOutputs;
        int i, j;
        for(numOfSamp=0; numOfSamp<NumberOfSamples; ++numOfSamp)
        {
                fprintf(outfp,"\n");
                for(numOfInputs=0; numOfInputs<numinVar; ++numOfInputs)
            {
                    fprintf(outfp,"inVar[%2d][%d] = %7.3lf ", numOfSamp+1
                        , numOfInputs+1, InputData[numOfSamp][numOfInputs]);
                }
                fprintf(outfp,"\t");
                for(numOfOutputs=0; numOfOutputs<numOutVar; ++numOfOutputs)
            {
                    fprintf(outfp,"outVar[%2d][%d] = %6.3lf ", numOfSamp+1,
                        numOfOutputs+1, OutputData[numOfSamp][numOfOutputs]);
                }
        }
}

int getInVarIndex(C, name, num)
struct controller *C;
char *name;
int num;
{
        int i;
```

```
            for(i=0; i < C->numOfInputVar; i++)
            {
                    if(!strcmp(name, C->inputVars[i].varName))
                            return(i);
            }
            fprintf(stderr, "\ngetInVarIndex: Error in rule[%d]! ",num);
            fprintf(stderr, "Variable '%s' does not exist!\n\n", name);
            exit(-1);
}


int getInMemIndex(C, name, num, varIndex)
struct controller *C;
char *name;
int num;
int varIndex;
{
            int i;
            for(i=0; i < C->inputVars[varIndex].numOfMembers; i++)
            {
                    if(!strcmp(name, C->inputVars[varIndex].member[i].memName))
                    return(i);
            }
            fprintf(stderr, "\ngetInMemIndex: Error in rule[%d]! ",num);
            fprintf(stderr, "Member '%s' does not exist!\n\n", name);
            exit(-1);
}


int getOutVarIndex(C, name, num)
struct controller *C;
char *name;
int num;
{
            int i;
            for(i=0; i < C->numOfOutputVar; i++)
            {
                    if(!strcmp(name, C->outputVars[i].varName))
                            return(i);
            }
            fprintf(stderr, "\ngetOutVarIndex: Error in rule[%d]! ",num);
            fprintf(stderr, "Variable '%s' does not exist!\n\n", name);
            exit(-1);
}


int getOutMemIndex(C, name, num, varIndex)
struct controller *C;
char *name;
int num;
int varIndex;
{
            int i;
            for(i=0; i < C->outputVars[varIndex].numOfMembers; i++)
            {
                    if(!strcmp(name, C->outputVars[varIndex].member[i].memName))
                    return(i);
            }
            fprintf(stderr, "\ngetOutMemIndex: Error in rule[%d]! ",num);
            fprintf(stderr, "Member '%s' does not exist!\n\n", name);
            exit(-1);
}
```

```c
int buildRuleBase(C,ruleP,fpGR,num,symbol)
struct controller *C;
struct ruleComp *ruleP;
FILE **fpGR;
int num;
char *symbol;
{
        char symbolIS[MAX];
        if(!strcmp(symbol, "IF"))
        {
                ruleP->type = IF;
                fscanf(*fpGR,"%s", ruleP->varName);
                ruleP->varIndex = getInVarIndex(C, ruleP->varName, num);
                fscanf(*fpGR,"%s", symbolIS);
                if(strcmp(symbolIS, "IS"))
                {
                    fprintf(stderr, "\nbuildRuleBase: Error in rule %d",num);
                        fprintf(stderr, ". Expecting IS!\n");
                        exit(-1);
                }
                fscanf(*fpGR,"%s", ruleP->memName);
                ruleP->memIndex=getInMemIndex(C, ruleP->memName ,
                                                num, ruleP->varIndex);
                return(1);
        }else if(!strcmp(symbol, "AND")){
                ruleP->type = AND;
                fscanf(*fpGR,"%s", ruleP->varName);
                ruleP->varIndex = getInVarIndex(C, ruleP->varName, num);
                fscanf(*fpGR,"%s", symbolIS);
                if(strcmp(symbolIS, "IS"))
                {
                    fprintf(stderr, "\nbuildRuleBase: Error in rule %d",num);
                        fprintf(stderr, ". Expecting IS!\n");
                        exit(-1);
                }
                fscanf(*fpGR,"%s", ruleP->memName);
                ruleP->memIndex=getInMemIndex(C, ruleP->memName ,
                                                num, ruleP->varIndex);
                return(1);
        }else if(!strcmp(symbol, "OR")){
                ruleP->type = OR;
                fscanf(*fpGR,"%s", ruleP->varName);
                ruleP->varIndex = getInVarIndex(C, ruleP->varName, num);
                fscanf(*fpGR,"%s", symbolIS);
                if(strcmp(symbolIS, "IS"))
                {
                    fprintf(stderr, "\nbuildRuleBase: Error in rule %d",num);
                        fprintf(stderr, ". Expecting IS!\n");
                        exit(-1);
                }
                fscanf(*fpGR,"%s", ruleP->memName);
                ruleP->memIndex=getInMemIndex(C, ruleP->memName ,
                                                num, ruleP->varIndex);
                return(1);
        }else if(!strcmp(symbol, "THEN")){
                ruleP->type = THEN;
                fscanf(*fpGR,"%s", ruleP->varName);
                ruleP->varIndex = getOutVarIndex(C, ruleP->varName, num);
                fscanf(*fpGR,"%s", symbolIS);
```

```
                        if(strcmp(symbolIS, "IS"))
                        {
                                fprintf(stderr, "\nbuildRuleBase: Error in rule %d",num);
                                        fprintf(stderr, ". Expecting IS!\n");
                                        exit(-1);
                        }
                        fscanf(*fpGR,"%s", ruleP->memName);
                        ruleP->memIndex = getOutMemIndex(C, ruleP->memName,
                                                        num, ruleP->varIndex);
                        return(0);
                }else{
                    fprintf(stderr, "\nbuildRuleBase: Error in rule %d file. ", num);
                                fprintf(stderr, "Check format!");
                                exit(-1);
                }
}


struct ruleComp *allocateComp()
{
        struct ruleComp *RC;
        if((RC = (struct ruleComp *)
                        malloc(sizeof(struct ruleComp)))==NULL)
        {
                fprintf(stderr,"\nallocateComp: not enough memory for rules.\n");
                        exit(-1);
        }
        return(RC);
}


getRules(C,R)
/* This function is used to get the rule from RuleFile. */
struct controller *C;
struct ruleObject *R;
{
        FILE            .       *fpGetRules;
        int                     numOfRules;
        char                    symbol[MAX];
        int                     endFlag;
        int                     newNode;
        struct ruleComp *P;
        if(numfiles == 0) {
        fprintf(outfp," Enter the name of the rule file. ");
                        fprintf(outfp,"-----------> ");
        }
        fscanf(infp,"%s",RuleFile);
        if((fpGetRules=fopen(RuleFile,"r")) == NULL)
        {
                fprintf(stderr,"Could not open %s\n",RuleFile);
                exit(-1);
        }
        fscanf(fpGetRules,"%d",&(R->numOfRules));
        if((R->rule = (struct ruleComp *)
                        malloc((R->numOfRules)*sizeof(struct ruleComp)))==NULL)
        {
                fprintf(stderr,"\ngetRules: not enough memory for rules.\n");
                exit(-1);
        }
        for(numOfRules=0; numOfRules<R->numOfRules; ++numOfRules)
    {
                endFlag = 0;
```

```c
                newNode = 0;
                P = &(R->rule[numOfRules]);
                while( !feof(fpGetRules) && !endFlag)
                {
                        fscanf(fpGetRules,"%s", symbol);
                        if(!strcmp(symbol, "END"))
                        {
                                endFlag = 1;
                        }else{
                                newNode=buildRuleBase(C,P,&fpGetRules,
                                                numOfRules, symbol);
                                if(newNode)
                                {
                                        P->next=allocateComp();
                                        P= P->next;
                                }else{
                                        P->next=NULL;
                                }
                        }
                }
                printf("\n");
        }

        fclose(fpGetRules);/*Close the file */
}/* End of function getRules. */

void printRules(R)
struct ruleObject *R;
{
        struct ruleComp *P;
        int             numOfRules;
        fprintf(stdout,"\n\n");
        for(numOfRules=0; numOfRules<R->numOfRules; ++numOfRules)
        {
                P = &(R->rule[numOfRules]);
                fprintf(stdout,"RULE[%d] = ",numOfRules);
                while (P!=NULL)
                {
                        switch(P->type)
                        {
                                case IF:
                                        fprintf(stdout,"IF ");
                                        break;
                                case AND:
                                        fprintf(stdout,"AND ");
                                        break;
                                case OR:
                                        fprintf(stdout,"OR ");
                                        break;
                                case THEN:
                                        fprintf(stdout,"THEN ");
                                        break;
                                default:
                                fprintf(stderr,"\nprintRules: ");
                                        fprintf(stderr,"In case.\n");
                                exit(-1);
                        }
                        fprintf(stdout,"%s IS %s ", P->varName, P->memName);
                        P=P->next;
                }
        }
```

```c
                fprintf(stdout,"END\n");
        }
        fprintf(stdout,"\n\n");
}


void freeRules(R)
struct ruleObject *R;
{
        struct ruleComp *P;
        struct ruleComp *tmp;
        int numOfRules;
        for(numOfRules=0; numOfRules<R->numOfRules; ++numOfRules)
    {
                P = &(R->rule[numOfRules]);
                while (P->next!=NULL)
                {
                        tmp = P->next;
                        P->next=P->next->next;
                        free(tmp);
                }
        }
        free(R->rule);
}




void evalMembers(memberOne, memberTwo)
/* This function is used to evaluate the validity of two member functions */
struct memberObject *memberOne;
struct memberObject *memberTwo;
{
 switch(memberOne->type)
 {
        case TRIANGULAR:
            switch(memberTwo->type)
            {
                case TRIANGULAR:
                        break;
                case S_FUNCTION:
                        fprintf(stderr,"\nevalMembers: Error invalid ");
                        fprintf(stderr,"combination of member functions!");
                        fprintf(stderr,"\n\n\tS_FUNCTION ... TRIANGULAR \n");
                        exit(1);
                        break;
                case Z_FUNCTION:
                        break;
                default:
                fprintf(stderr,"\nevalMembers: Error in inner case");
                        exit(1);
            }
          break;
        case S_FUNCTION:
            switch(memberTwo->type)
            {
                case TRIANGULAR:
                        break;
                case S_FUNCTION:
                        fprintf(stderr,"\nevalMembers: Error invalid ");
                        fprintf(stderr,"combination of member functions!");
                        fprintf(stderr,"\n\n\tS_FUNCTION ... S_FUNCTION\n");
```

```c
                                        exit(1);
                                        break;
                        case Z_FUNCTION:
                                        break;
                        default:
                        fprintf(stderr,"\nevalMembers: Error in inner case");
                                        exit(1);
                }
                break;
        case Z_FUNCTION:
            switch(memberTwo->type)
                {
                        case TRIANGULAR:
                                        fprintf(stderr,"\nevalMembers: Error invalid ");
                                        fprintf(stderr,"combination of member functions!");
                                        fprintf(stderr,"\n\n\tTRIANGULAR ... Z_FUNCTION\n");
                                        exit(1);
                                        break;
                        case S_FUNCTION:
                                        fprintf(stderr,"\nevalMembers: Error invalid ");
                                        fprintf(stderr,"combination of member functions!");
                                        fprintf(stderr,"\n\n\tS_FUNCTION ... Z_FUNCTION\n");
                                        exit(1);
                                        break;
                        case Z_FUNCTION:
                                        fprintf(stderr,"\nevalMembers. Error invalid ");
                                        fprintf(stderr,"combination of member functions!");
                                        fprintf(stderr,"\n\n\tZ_FUNCTION ... Z_FUNCTION\n");
                                        exit(1);
                                        break;
                        default:
                        fprintf(stderr,"\nevalMembers: Error in inner case");
                                        exit(1);
                }
                break;
        default:
        fprintf(stderr,"\nevalMembers: Error in outer case");
                        exit(1);
    }
}

void checkMem(TheVar)
/* This function will be used to determine the number the of criteria used  */
/* to calculate the fitness */
struct varObject *TheVar;
{
        int j,k;
        for(j=0; j < TheVar->numOfMembers; j++)
        {
                for(k=j-1; k>=0; k--)
                {
                        evalMembers(TheVar->member[j],
                                TheVar->member[k]);
                }
        }
}

int getType(nameM)
char *nameM;
{
```

```c
        int choice;              /* varible that holds the        */
                                 /* choice of the user.                      */

/*
*       This part of the program will print the list of choices for the user.
*/
        if(numfiles == 0){
                /*fprintf(outfp,"\n");*/
                fprintf(outfp,"\tMember function '%s' is of what type:\n\n",
                        nameM);
                fprintf(outfp,"\t\t1.  Triangular.\n");
                fprintf(outfp,"\t\t2.  S-Function.\n");
                fprintf(outfp,"\t\t3.  Z-Function.\n");
                fprintf(outfp,"\n");
                fprintf(outfp,"\tEnter your choice: ");
        }
                fscanf(infp,"%d",&choice);
        if(numfiles == 0)
                fprintf(outfp,"\n");
/*
*       This part of the program will return the value for the function
*       that corresponds to the users choice.
*/
        if (choice == 1){
                return(TRIANGULAR);       /* Triangular */
        } else if (choice == 2){
                return(S_FUNCTION);       /* S-Function */
        }else if (choice == 3){
                return(Z_FUNCTION);       /* Z-Function */
        }else {
                fprintf(stderr,"\nIncorrect entry for type!");
                exit(-1);
        }         }


void allocateInputVar(C)
/* This function is used to allocate space for the input variables */
struct controller *C;
{
        int i,j;               /* for loop variables */
        /* get the number of input variables */
        if(numfiles == 0){
        fprintf(outfp,"\n Enter number of input variables ");
                fprintf(outfp,"-------------> ");
        }
        fscanf(infp,"%d",&(C->numOfInputVar));
        if((C->inputVars = (struct varObject *)
                malloc((C->numOfInputVar)*sizeof(struct varObject)))==NULL)
        {
                fprintf(stderr,"not enough memory for %s.\n","inputVars");
                exit(-1);
        }
        for(i=0; i < C->numOfInputVar; i++)
        {
                if(numfiles == 0){
                fprintf(outfp,"\n Enter name of input variable");
                        fprintf(outfp,"[%d] ", i+1);
                        fprintf(outfp,"-------------> ");
                }
                fscanf(infp,"%s",C->inputVars[i].varName);
```

```c
if(numfiles == 0){
        fprintf(outfp,"\n Enter the lower bound ");
        fprintf(outfp,"for the variable ");
        fprintf(outfp,"'%s' --> ", C->inputVars[i].varName);
}
fscanf(infp,"%lf",&(C->inputVars[i].lowerBound));
if(numfiles == 0){
fprintf(outfp,"\n Enter the upper bound ");
        fprintf(outfp,"for the variable ");
        fprintf(outfp,"'%s' --> ", C->inputVars[i].varName);
}
fscanf(infp,"%lf",&(C->inputVars[i].upperBound));
if( C->inputVars[i].upperBound < C->inputVars[i].lowerBound)
{
        fprintf(stderr,"\n Upper bound is greater than ");
        fprintf(stderr,"lower bound of variable ");
        fprintf(stderr,"'%s'.\n", C->inputVars[i].varName);
        exit(-1);
}
C->inputVars[i].delta = C->inputVars[i].upperBound -
                C->inputVars[i].lowerBound;
if(numfiles == 0){
        fprintf(outfp,"\n Enter the number of bits ");
        fprintf(outfp,"for the variable ");
        fprintf(outfp,"'%s' --> ", C->inputVars[i].varName);
}
fscanf(infp,"%d",&(C->inputVars[i].numBits));
if(numfiles == 0){
        fprintf(outfp,"\n Enter number of member functions ");
        fprintf(outfp,"for variable ");
        fprintf(outfp,"'%s' --> ", C->inputVars[i].varName);
}
fscanf(infp,"%d",&(C->inputVars[i].numOfMembers));
/* allocate space for member functions for the variable */
if((C->inputVars[i].member = (struct memberObject *)
        malloc((C->inputVars[i].numOfMembers) *
        sizeof(struct memberObject)))==NULL)
{
        fprintf(stderr,"not enough memory for the ");
        fprintf(stderr,"member functions of variable ");
        fprintf(stderr,"'%s'.\n", C->inputVars[i].varName);
        exit(-1);
}
if(numfiles == 0)
fprintf(outfp,"\n");
/* get name and type for member functions */
for(j=0; j < C->inputVars[i].numOfMembers; j++)
{
        if(numfiles == 0){
        fprintf(outfp,"\tEnter name of member ");
                fprintf(outfp,"function %2d ", j+1);
                fprintf(outfp," --> ");
        }
        fscanf(infp,"%s",C->inputVars[i].member[j].memName);
        C->inputVars[i].member[j].type =
                getType(C->inputVars[i].member[j].memName);
}
/* check for invalid combination of member functions */
checkMem(C->inputVars[i]);
}
```

```
}

void freeInputVar(C)
/* This function is used to deallocate space for the input variables */
struct controller *C;
{
        int i;
        for(i=0; i < C->numOfInputVar; i++)
        {
                free(C->inputVars[i].member);
        }
        free(C->inputVars);
}


void allocateOutputVar(C)
/* This function is used to allocate space for the output variables */
struct controller *C;
{
        int i,j;                /* for loop variables */
        /* get the number of output variables */
        if(numfiles == 0){
        fprintf(outfp,"\n Enter number of output variables ");
                fprintf(outfp,"----------> ");
        }
        fscanf(infp,"%d",&(C->numOfOutputVar));
        if((C->outputVars = (struct varObject *)
                malloc((C->numOfOutputVar)*sizeof(struct varObject)))==NULL)
        {
                fprintf(stderr,"not enough memory for %s.\n","outputVars");
                exit(-1);
        }
        for(i=0; i < C->numOfOutputVar; i++)
        {
                if(numfiles == 0){
                fprintf(outfp,"\n Enter name of output variable");
                        fprintf(outfp,"[%d] ", i+1);
                        fprintf(outfp,"----------> ");
                }
                fscanf(infp,"%s",C->outputVars[i].varName);
                if(numfiles == 0){
                        fprintf(outfp,"\n Enter the lower bound  ");
                        fprintf(outfp,"for the variable ");
                        fprintf(outfp,"'%s' --> ", C->outputVars[i].varName);
                }
                fscanf(infp,"%lf",&(C->outputVars[i].lowerBound));
                if(numfiles == 0){
                        fprintf(outfp,"\n Enter the upper bound  ");
                        fprintf(outfp,"for the variable ");
                        fprintf(outfp,"'%s' --> ", C->outputVars[i].varName);
                }
                fscanf(infp,"%lf",&(C->outputVars[i].upperBound));
                if( C->outputVars[i].upperBound < C->outputVars[i].lowerBound)
                {
                        fprintf(stderr,"\n Upper bound is greater than ");
                        fprintf(stderr,"lower bound of variable ");
                        fprintf(stderr,"'%s'.\n", C->outputVars[i].varName);
                        exit(-1);
                }
                C->outputVars[i].delta = C->outputVars[i].upperBound -
                                C->outputVars[i].lowerBound;
```

```c
                if(numfiles == 0){
                        fprintf(outfp,"\n Enter the number of bits ");
                        fprintf(outfp,"for the variable ");
                        fprintf(outfp,"'%s' --> ", C->outputVars[i].varName);
                }
                fscanf(infp,"%d",&(C->outputVars[i].numBits));
                if(numfiles == 0){
                        fprintf(outfp,"\n Enter number of member functions ");
                        fprintf(outfp,"for variable ");
                        fprintf(outfp,"'%s' --> ", C->outputVars[i].varName);
                }
                fscanf(infp,"%d",&(C->outputVars[i].numOfMembers));
                /* allocate space for member functions for the variable */
                if((C->outputVars[i].member = (struct memberObject *)
                        malloc((C->outputVars[i].numOfMembers) *
                        sizeof(struct memberObject)))==NULL)
                {
                        fprintf(stderr,"not enough memory for the ");
                        fprintf(stderr,"member functions of variable ");
                        fprintf(stderr,"'%s'.\n",C->outputVars[i].varName);
                        exit(-1);
                }
                if(numfiles == 0)
                fprintf(outfp,"\n");
                /* get name and type for member functions */
                for(j=0; j < C->outputVars[i].numOfMembers; j++)
                {
                        if(numfiles == 0){
                        fprintf(outfp,"\tEnter name of member ");
                                fprintf(outfp,"function %2d ", j+1);
                                fprintf(outfp," --> ");
                        }
                        fscanf(infp,"%s", C->outputVars[i].member[j].memName);
                        C->outputVars[i].member[j].type =
                                getType(C->outputVars[i].member[j].memName);
                }
                /* check for invalid combination of member functions */
                checkMem(C->outputVars[i]);
        }
}


void freeOutputVar(C)
/* This function is used to deallocate space for the output variables */
struct controller *C;
{
        int i;
        for(i=0; i < C->numOfOutputVar; i++)
        {
                free(C->outputVars[i].member);
        }
        free(C->outputVars);
}


void defineController(C)
/* This function is used to allocate space for the variables */
struct controller *C;
{
        if(numfiles == 0){
        fprintf(outfp,"\n Enter name of controller ");
                fprintf(outfp,"----------------> ");
```

```c
        }
                        fscanf(infp,"%s",C->controllerName);
                allocateInputVar(C);
                allocateOutputVar(C);
}


void freeVariables(C)
/* This function is used to allocate space for the variables */
struct controller *C;
{
                freeInputVar(C);
                freeOutputVar(C);
}


void printMemType(Type)
int Type;
{
                switch(Type)
                {
                        case TRIANGULAR:
                                fprintf(outfp,"\n\tTiangular: ");
                                break;
                        case S_FUNCTION:
                                fprintf(outfp,"\n\tS-Function:");
                                break;
                        case Z_FUNCTION:
                                fprintf(outfp,"\n\tZ-Function:");
                                break;
                        default:
                                fprintf(stderr,"\nprintMemType: Error in the case");
                                exit(1);
                }
}

void printValues(member)
/* This function is used to print the values of the verticies */
struct memberObject *member;
{
                fprintf(outfp,"\n");
                switch(member->type)
                {

                        case TRIANGULAR:
                                fprintf(outfp,"\n\t\t\t\tleftVertex   = %.3lf",
                                                        member->leftVertex);
                                fprintf(outfp,"\n\t\t\t\tcenterVertex = %.3lf",
                                                        member->centerVertex);
                                fprintf(outfp,"\n\t\t\t\trightVertex  = %.3lf",
                                                        member->rightVertex);
                                break;
                        case S_FUNCTION:
                                fprintf(outfp,"\n\t\t\t\tleftVertex   = %.3lf",
                                                        member->leftVertex);
                                fprintf(outfp,"\n\t\t\t\tcenterVertex = %.3lf",
                                                        member->centerVertex);
                                break;
                        case Z_FUNCTION:
                                fprintf(outfp,"\n\t\t\t\tcenterVertex = %.3lf",
```

58

```c
                                                member->centerVertex);
                        fprintf(outfp,"\n\t\t\t\trightVertex  = %.3lf",
                                                member->rightVertex);
                        break;
                default:
                        fprintf(stderr,"\nprintValues: Error in the case");
                        exit(1);
        }
        fprintf(outfp,"\n");
}


void printInVars(C, pValue)
/* this function is used to check the input variables */
struct controller *C;
int pValue;
{
        int i,j;
        fprintf(outfp,"\n\n————— INPUTS —————\n");
        fprintf(outfp,"\nnumber of input varibles = %d", C->numOfInputVar);
        for(i=0; i < C->numOfInputVar; i++)
        {
                fprintf(outfp,"\n\nvariable[%d] = '%s' and it has ",
                                i+1, C->inputVars[i].varName);
                fprintf(outfp,"%d members:", C->inputVars[i].numOfMembers);
                fprintf(outfp,"\n");
                for(j=0; j < C->inputVars[i].numOfMembers; j++)
                {
                        printMemType(C->inputVars[i].member[j].type);
                        fprintf(outfp,"\tmember[%d] = ", j+1);
                        fprintf(outfp,"%s ",C->inputVars[i].member[j].memName);
                        if(pValue==1)
                                printValues(C->inputVars[i].member[j]);
                }
                fprintf(outfp,"\n\nlower bounds = %lf",
                                                C->inputVars[i].lowerBound);
                fprintf(outfp,"\nupper bounds = %lf",
                                                C->inputVars[i].upperBound);
                fprintf(outfp,"\ndelta        = %lf",
                                                C->inputVars[i].delta);
                fprintf(outfp,"\nnum. of bits = %d",
                                                C->inputVars[i].numBits);
        }
        fprintf(outfp,"\n\n");
}


void printOutVars(C, pValue)
/* this function is used to check the output variables */
struct controller *C;
int pValue;
{
        int i,j;
        fprintf(outfp,"\n\n————— OUTPUTS —————\n");
        fprintf(outfp,"\nnumber of output varibles = %d", C->numOfOutputVar);
        for(i=0; i < C->numOfOutputVar; i++)
        {
                fprintf(outfp,"\n\nvariable[%d] = '%s' and it has ",
                                i+1, C->outputVars[i].varName);
                fprintf(outfp,"%d members:", C->outputVars[i].numOfMembers);
                fprintf(outfp,"\n");
                for(j=0; j < C->outputVars[i].numOfMembers; j++)
```

59

```c
                {
                        printMemType(C->outputVars[i].member[j].type);
                        fprintf(outfp,"\tmember[%d] = ", j+1);
                        fprintf(outfp,"%s ",C->outputVars[i].member[j].memName);
                        if(pValue==1)
                                printValues(C->outputVars[i].member[j]);
                }
                fprintf(outfp,"\n\nlower bounds = %lf",
                                                C->outputVars[i].lowerBound);
                fprintf(outfp,"\nupper bounds = %lf",
                                                C->outputVars[i].upperBound);
                fprintf(outfp,"\ndelta       = %lf",
                                                C->outputVars[i].delta);
                fprintf(outfp,"\nnum. of bits = %d",
                                                C->outputVars[i].numBits);
        }
        fprintf(outfp,"\n\n");
}


void printVariables(C , P)
/* this function is used to print all the variables */
struct controller *C;
int P;
{
        fprintf(outfp,"\n\n\tCONTROLLER: ");
        fprintf(outfp,"%s", C->controllerName);
        printInVars(C, P);
        printOutVars(C, P);
}

void evalMemberCrite(member, count, valid)
/* This function is used to evaluate the criteria for a single member */
struct memberObject *member;
double *count;
double *valid;
{
        switch(member->type)
        {
                case TRIANGULAR:
                        *count = *count + 3.0;
                        if(member->centerVertex >= member->leftVertex)
                                *valid = *valid + 1.0;
                        if(member->rightVertex > member->leftVertex)
                                *valid = *valid + 1.0;
                        if(member->rightVertex >= member->centerVertex)
                                *valid = *valid + 1.0;
                        break;
                case S_FUNCTION:
                        *count = *count + 1.0;
                        if(member->centerVertex >= member->leftVertex)
                                *valid = *valid + 1.0;
                        break;
                case Z_FUNCTION:
                        *count = *count + 1.0;
                        if(member->rightVertex >= member->centerVertex)
                                *valid = *valid + 1.0;
                        break;
                default:
                        fprintf(stderr,"\nevalMemberCrite: Error in the case");
```

```c
                                    exit(1);
                }
}

void compareMemberCrite(memberOne, memberTwo, count, valid)
/* This function is used to evaluate the criteria of two member */
struct memberObject *memberOne;
struct memberObject *memberTwo;
double *count;
double *valid;
{
  switch(memberOne->type)
  {
        case TRIANGULAR:
            switch(memberTwo->type)
            {
                case TRIANGULAR:
                        *count = *count + 6.0;
                        if(memberOne->leftVertex >= memberTwo->leftVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->centerVertex > memberTwo->leftVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->centerVertex > memberTwo->centerVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->rightVertex > memberTwo->leftVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->rightVertex > memberTwo->centerVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->rightVertex >= memberTwo->rightVertex)
                                *valid = *valid + 1.0;

                        break;
                case S_FUNCTION:
                        fprintf(stderr,"\ncompareMemberCrite: Error invalid ");
                        fprintf(stderr,"combination of member functions!");
                        fprintf(stderr,"\n\n\tS_FUNCTION ... TRIANGULAR \n");
                        exit(1);
                        break;
                case Z_FUNCTION:
                        *count = *count + 3.0;
                        if(memberOne->centerVertex > memberTwo->centerVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->rightVertex > memberTwo->centerVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->rightVertex >= memberTwo->rightVertex)
                                *valid = *valid + 1.0;

                        break;
                default:
                fprintf(stderr,"\ncompareMemberCrite: Error in inner case");
                        exit(1);
            }
            break;
        case S_FUNCTION:
            switch(memberTwo->type)
            {
                case TRIANGULAR:
                        *count = *count + 3.0;
                        if(memberOne->leftVertex >= memberTwo->leftVertex)
                                *valid = *valid + 1.0;
                        if(memberOne->centerVertex > memberTwo->leftVertex)
                                *valid = *valid + 1.0;
```

```c
                        if(memberOne->centerVertex > memberTwo->centerVertex)
                                *valid = *valid + 1.0;
                        break;
                case S_FUNCTION:
                        fprintf(stderr,"\ncompareMemberCrite: Error invalid ");
                        fprintf(stderr,"combination of member functions!");
                        fprintf(stderr,"\n\n\tS_FUNCTION ... S_FUNCTION\n");
                        exit(1);
                        break;
                case Z_FUNCTION:
                        *count = *count + 1.0;
                        if(memberOne->centerVertex > memberTwo->centerVertex)
                                *valid = *valid + 1.0;
                        break;
                default:
                fprintf(stderr,"\ncompareMemberCrite: Error in inner case");
                        exit(1);
        }
        break;
case Z_FUNCTION:
        switch(memberTwo->type)
        {
                case TRIANGULAR:
                        fprintf(stderr,"\ncompareMemberCrite: Error invalid ");
                        fprintf(stderr,"combination of member functions!");
                        fprintf(stderr,"\n\n\tTRIANGULAR ... Z_FUNCTION\n");
                        exit(1);
                        break;
                case S_FUNCTION:
                        fprintf(stderr,"\ncompareMemberCrite: Error invalid ");
                        fprintf(stderr,"combination of member functions!");
                        fprintf(stderr,"\n\n\tS_FUNCTION ... Z_FUNCTION\n");
                        exit(1);
                        break;
                case Z_FUNCTION:
                        fprintf(stderr,"\ncompareMemberCrite: Error invalid ");
                        fprintf(stderr,"combination of member functions!");
                        fprintf(stderr,"\n\n\tZ_FUNCTION ... Z_FUNCTION\n");
                        exit(1);
                        break;
                default:
                fprintf(stderr,"\ncompareMemberCrite: Error in inner case");
                        exit(1);
        }
        break;
default:
fprintf(stderr,"\ncompareMemberCrite: Error in outer case");
        exit(1);
    }
}

evaluateCrite(TheVar, count, valid)
/* This function will be used to determine the number the of criteria used */
/* to calculate the fitness */
struct varObject *TheVar;
double *count;
double *valid;
{
        int j,k;
        for(j=0; j < TheVar->numOfMembers; j++)
```

```
                {.
                        evalMemberCrite(TheVar->member[j], count, valid);
                        for(k=j-1; k>=0; k--)
                        {
                                compareMemberCrite(TheVar->member[j],
                                        TheVar->member[k], count, valid);
                        }
                }
        }
}


void evalInMemCrite(C, count, valid)
/* This function will be used to determine the number the of criteria used */
/* to calculate the fitness */
struct controller *C;
double *count;
double *valid;
{
        int i;
        for(i=0; i < C->numOfInputVar; i++)
        {
                evaluateCrite(C->inputVars[i], count, valid);
        }
}


void evalOutMemCrite(C, count, valid)
/* This function will be used to determine the number the of criteria used */
/* to calculate the fitness */
struct controller *C;
double *count;
double *valid;
{
        int i;
        for(i=0; i < C->numOfOutputVar; i++)
        {
                evaluateCrite(C->outputVars[i], count, valid);
        }
}


double evalMemFitness(C)
/* This function will be used to calculate the fitness of the membership */
/* functions. The fitness is determined by the criteria for valid members */
struct controller *C;
{
        double ruleCount, validCrite;
        ruleCount = 0.0;
        validCrite = 0.0;
        evalInMemCrite(C, &ruleCount, &validCrite);
        evalOutMemCrite(C, &ruleCount, &validCrite);
/*
        fprintf(stdout, "\nThe number of criteria   = %.lf",ruleCount);
        fprintf(stdout, "\nCrite that were valid = %.lf", validCrite);
*/
        return(validCrite/ruleCount);
}
void countMembVert(member, count)
/* This function is used to count the number of vertecies for a single member */
struct memberObject *member;
int *count;
{
        switch(member->type)
```

```c
        {
                case TRIANGULAR:
                        *count = *count + 3;
                        break;
                case S_FUNCTION:
                        *count = *count + 2;
                        break;
                case Z_FUNCTION:
                        *count = *count + 2;
                        break;
                default:
                        fprintf(stderr,"\ncountMembVert: Error in the case");
                        exit(1);
        }
}


void countVarBit(TheVar, bitCount)
/* This function will be used to determine the number the of bits of a */
/* a single variable */
struct varObject *TheVar;
int *bitCount;
{
        int j;
        int vertCount;
        vertCount = 0;
        for(j=0; j < TheVar->numOfMembers; j++)
        {
                countMembVert(TheVar->member[j], &vertCount);
        }
        *bitCount = *bitCount + (vertCount*(TheVar->numBits));
}


int countBits(C)
/* This function will be used to determine the number of bits needed */
struct controller *C;
{
        int bitCount, i, j;
        bitCount = 0;
        for(j=0; j < C->numOfInputVar; j++)
        {
                countVarBit(C->inputVars[j], &bitCount);
        }
        for(i=0; i < C->numOfOutputVar; i++)
        {
                countVarBit(C->outputVars[i], &bitCount);
        }
        fprintf(stdout, "\nThe number of bits = %d.\n", bitCount);
        return(bitCount);
}


vertexType getRealNumber(critter,start,stop,numBits,lowerBound,delta)
/* This function is used to get real numbers from the chrome */
unsigned *critter;
int *start;
int *stop;
int numBits;
vertexType lowerBound;
vertexType delta;
{
        vertexType realValue;
```

```c
/* section of chromosome containing current integer field */
*start = *stop + 1;
*stop  = *start + numBits - 1;
    /* printf("\n start = %d , stop = %d lchrom = %d", *start, *stop,
                                lchrom); */
/* check if enough bits remain, if not, exit program */
if(*stop > lchrom)
        {
                fprintf(stderr,"\nError in getRealNumber function.\n");
                exit(-1);
        }
        realValue =((double)ithruj2int(*start, *stop, critter))/
                        pow(2.0, (double)(numBits));
        realValue = realValue*delta + lowerBound;
        return(realValue);
}


void getInValues(critter, C, start, stop)
/* This function is used to get values for the vertecies of */
/* the input variables */
unsigned *critter;
struct controller *C;
int *start;
int *stop;
{
        int i,j;
        for(i=0; i < C->numOfInputVar; i++)
        {
          for(j=0; j < C->inputVars[i].numOfMembers; j++)
          {
                switch(C->inputVars[i].member[j].type)
                {
                case TRIANGULAR:
                  C->inputVars[i].member[j].leftVertex =
                    getRealNumber(critter,start,stop,C->inputVars[i].numBits,
                        C->inputVars[i].lowerBound, C->inputVars[i].delta);
                  C->inputVars[i].member[j].centerVertex =
                    getRealNumber(critter,start,stop,C->inputVars[i].numBits,
                        C->inputVars[i].lowerBound, C->inputVars[i].delta);
                  C->inputVars[i].member[j].rightVertex =
                    getRealNumber(critter,start,stop,C->inputVars[i].numBits,
                        C->inputVars[i].lowerBound, C->inputVars[i].delta);
                  break;
                case S_FUNCTION:
                  C->inputVars[i].member[j].leftVertex =
                    getRealNumber(critter,start,stop,C->inputVars[i].numBits,
                        C->inputVars[i].lowerBound, C->inputVars[i].delta);
                  C->inputVars[i].member[j].centerVertex =
                    getRealNumber(critter,start,stop,C->inputVars[i].numBits,
                        C->inputVars[i].lowerBound, C->inputVars[i].delta);
                  break;
                case Z_FUNCTION:
                  C->inputVars[i].member[j].centerVertex =
                    getRealNumber(critter,start,stop,C->inputVars[i].numBits,
                        C->inputVars[i].lowerBound, C->inputVars[i].delta);
                  C->inputVars[i].member[j].rightVertex =
                    getRealNumber(critter,start,stop,C->inputVars[i].numBits,
                        C->inputVars[i].lowerBound, C->inputVars[i].delta);
                  break;
                default:
```

```c
void getValues(critter, C)
/* This function is used to get the values for the vertecies. */
unsigned *critter;
struct controller *C;
{
        int start, stop;
        start = 0;
        stop = 0;
        getInValues(critter, C, &start, &stop);
        getOutValues(critter, C, &start, &stop);
}


void cloneInputVar(original, clone)
/* This function is used clone the input variables */
struct controller *original;
struct controller *clone;
{
        int i, j;
        for(i=0; i < clone->numOfInputVar; i++)
        {
                strcpy(clone->inputVars[i].varName,
                                        original->inputVars[i].varName);
                clone->inputVars[i].numOfMembers=
                                        original->inputVars[i].numOfMembers;
                clone->inputVars[i].lowerBound=
                                        original->inputVars[i].lowerBound;
                clone->inputVars[i].upperBound=
                                        original->inputVars[i].upperBound;
                clone->inputVars[i].delta=
                                        original->inputVars[i].delta;
                clone->inputVars[i].numBits=
                                        original->inputVars[i].numBits;
                if((clone->inputVars[i].member = (struct memberObject *)
                        malloc((clone->inputVars[i].numOfMembers) *
                        sizeof(struct memberObject)))==NULL)
                {
                        fprintf(stderr,"not enough memory for the clone's ");
                        fprintf(stderr,"member functions of variable ");
                        fprintf(stderr,"'%s'.\n", clone->inputVars[i].varName);
                        exit(-1);
                }
                for(j=0; j < clone->inputVars[i].numOfMembers; j++)
                {
                        strcpy(clone->inputVars[i].member[j].memName,
                                original->inputVars[i].member[j].memName);
                        clone->inputVars[i].member[j].type =
                                original->inputVars[i].member[j].type;
                }
        }
}
void cloneOutputVar(original, clone)
/* This function is used to clone the output variables */
struct controller *original;
struct controller *clone;
{
        int i, j;
        for(i=0; i < clone->numOfOutputVar; i++)
        {
                strcpy(clone->outputVars[i].varName,
                                        original->outputVars[i].varName);
```

```c
                        clone->outputVars[i].numOfMembers=
                                        original->outputVars[i].numOfMembers;
                        clone->outputVars[i].lowerBound=
                                        original->outputVars[i].lowerBound;
                        clone->outputVars[i].upperBound=
                                        original->outputVars[i].upperBound;
                        clone->outputVars[i].delta=
                                        original->outputVars[i].delta;
                        clone->outputVars[i].numBits=
                                        original->outputVars[i].numBits;
                        if((clone->outputVars[i].member = (struct memberObject *)
                                malloc((clone->outputVars[i].numOfMembers) *
                                sizeof(struct memberObject)))==NULL)
                        {
                                fprintf(stderr,"not enough memory for the clone's ");
                                fprintf(stderr,"member functions of variable ");
                                fprintf(stderr,"%s'.\n", clone->outputVars[i].varName);
                                exit(-1);
                        }
                        for(j=0; j < clone->outputVars[i].numOfMembers; j++)
                        {
                                strcpy(clone->outputVars[i].member[j].memName,
                                        original->outputVars[i].member[j].memName);
                                clone->outputVars[i].member[j].type =
                                        original->outputVars[i].member[j].type;

                        }
                }
        }
}


void cloneController(original, clone)
/* This function is used to clone a controller */
struct controller *original;
struct controller *clone;
{
        strcpy(clone->controllerName, original->controllerName);
        clone->numOfInputVar = original->numOfInputVar;
        clone->numOfOutputVar = original->numOfOutputVar;
        if((clone->inputVars = (struct varObject *)
                malloc((clone->numOfInputVar)*sizeof(struct varObject)))==NULL)
        {
                fprintf(stderr,"not enough memory for %s.\n","clone inputVars");
                exit(-1);
        }
        if((clone->outputVars = (struct varObject *)
                malloc((clone->numOfOutputVar)*sizeof(struct varObject)))==NULL)
        {
            fprintf(stderr,"not enough memory for %s.\n","clone outputVars");
                exit(-1);
        }
        cloneInputVar(original, clone);
        cloneOutputVar(original, clone);

}




void store_data(int Generation)
/* This function is used to store the data to weightFile. The data will be  */
/* used to test the results of the sga.                        */
{
        FILE *fpStoreData;
```

67

```
                int · vertex;
                char WeightFile[120];         /* File used to store the vertex for this  */
                                              /* generation.                          */
                sprintf(WeightFile,"%s.%d",VertexFileBase,Generation);
                if((fpStoreData=fopen(WeightFile,"w")) == NULL)
                {
                        printf("Could not open %s\n",WeightFile);
                        exit(-1);
                }
                fprintf(fpStoreData,"%s\n",DataFile);
                fprintf(fpStoreData,"%d\n",NumberOfSamples);
                fprintf(fpStoreData,"%lf\n", bestfit.fitness);
                fprintf(fpStoreData, "%d\n",Generation);
                fprintf(fpStoreData, "%d\n",maxgen);
                fprintf(fpStoreData,"%f\n", pcross);
                fprintf(fpStoreData,"%f\n", pmutation);
                fprintf(fpStoreData, "%d\n",popsize);
                fclose(fpStoreData);
}/* End of function store_data. */




application()
/* This routine should contain any application-dependent     */
/* computations that should be performed before each GA cycle */
/* called by main() */
{
}




app_data()
/* application dependent data input, called by init_data() */
/* ask your input questions here, and put output in global variables */
{
}




app_free()
/* This routine should free any memory allocated */
/* in the application-dependent routines, called by freeall() */
{
        int j;
        freeVariables(&person);     /* deallocate variable space */
        freeRules(&ruleBase);       /* deallocate rule space */
        /* deallocate space for data variables */
        for(j = 0; j < NumberOfSamples; j++)
        {
                free(InputData[j]);
                free(OutputData[j]);
                free(CalculatedOutput[j]);
        }
        free(InputData);
        free(OutputData);
        free(CalculatedOutput);
}

app_init()
/* Application dependent initialization routine called by initialize(). */
{
}
```

APPENDIX D
SAMPLE RUN OF THE GA TUNER

```
-------------------------------------------------
|     SGA-C (v1.1) - A Simple Genetic Algorithm    |
|     (c) David E. Goldberg 1986, All Rights Reserved    |
|        C version by Robert E. Smith, U. of Alabama      |
|     v1.1 modifications by Jeff Earickson, Boeing Company |
-------------------------------------------------
```

SGA Parameters

Total Population size   = 50
Chromosome length (lchrom)   = 210
Maximum # of generations (maxgen)   = 200
Crossover probability (pcross)   = 0.600000
Mutation probability (pmutation)   = 0.010000

RUN 1 of 1: GENERATION 4->200

Generation 4 Accumulated Statistics:
Total Crossovers = 80, Total Mutations = 529
min = 0.347222   max = 0.430556   avg = 0.402778   sum = 20.138889
Global Best Individual so far, Generation 4:
Fitness = 0.430556:

CONTROLLER: simplePend

-------------- INPUTS --------------

number of input varibles = 2

variable[1] = 'error' and it has 3 members:

    Z-Function: member[1] = negative

                 centerVertex = -0.850
                 rightVertex = 0.711

    Tiangular: member[2] = zero

                 leftVertex   = -0.938
                 centerVertex = 0.195
                 rightVertex = 0.771

    S-Function: member[3] = positive

                 leftVertex   = 0.557
                 centerVertex = 0.711

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

variable[2] = 'Derror' and it has 3 members:

    Z-Function: member[1] = negative

                centerVertex = 0.527
                rightVertex = -0.809

    Tiangular: member[2] = zero

                leftVertex   = -0.498
                centerVertex = -0.496
                rightVertex = 0.090

    S-Function: member[3] = positive

                leftVertex   = -0.029
                centerVertex = 0.947

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

--------------- OUTPUTS ---------------

number of output varibles = 1

variable[1] = 'current' and it has 3 members:

    Z-Function: member[1] = negative

                centerVertex = -0.240
                rightVertex = 0.586

    Tiangular: member[2] = zero

                leftVertex   = -0.967
                centerVertex = -0.848
                rightVertex = 0.029

    S-Function: member[3] = positive

                leftVertex   = -0.908
                centerVertex = -0.082

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

RUN 1 of 1: GENERATION 29->200

Generation 29 Accumulated Statistics:
Total Crossovers = 464, Total Mutations = 3085
min = 0.458333   max = 0.962314   avg = 0.795700   sum = 39.784985
Global Best Individual so far, Generation 28:
Fitness = 0.963756:

RUN 1 of 1: GENERATION 59->200

Generation 59 Accumulated Statistics:
Total Crossovers = 903, Total Mutations = 6188
min = 0.430556   max = 0.985419   avg = 0.808049   sum = 40.402465
Global Best Individual so far, Generation 59:
Fitness = 0.985419:

RUN 1 of 1: GENERATION 79->200

Generation 79 Accumulated Statistics:
Total Crossovers = 1205, Total Mutations = 8258
min = 0.444444   max = 0.989729   avg = 0.826472   sum = 41.323584
Global Best Individual so far, Generation 78:
Fitness = 0.989907:

RUN 1 of 1: GENERATION 99->200

Generation 99 Accumulated Statistics:
Total Crossovers = 1509, Total Mutations = 10320
min = 0.430556   max = 0.991671   avg = 0.776286   sum = 38.814297
Global Best Individual so far, Generation 97:
Fitness = 0.991671:

CONTROLLER: simplePend

—————— INPUTS ——————

number of input varibles = 2

variable[1] = 'error' and it has 3 members:

Z-Function: member[1] = negative

71

centerVertex = -0.650
                                    rightVertex = 0.043

        Tiangular: member[2] = zero

                                    leftVertex   = -0.389
                                    centerVertex = 0.025
                                    rightVertex = 0.502

        S-Function: member[3] = positive

                                    leftVertex   = -0.012
                                    centerVertex = 0.379

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

variable[2] = 'Derror' and it has 3 members:

        Z-Function: member[1] = negative

                                    centerVertex = -0.273
                                    rightVertex = 0.018

        Tiangular: member[2] = zero

                                    leftVertex   = -0.451
                                    centerVertex = -0.199
                                    rightVertex = 0.502

        S-Function: member[3] = positive

                                    leftVertex   = -0.035
                                    centerVertex = 0.562

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

--------------- OUTPUTS ---------------

number of output varibles = 1

variable[1] = 'current' and it has 3 members:

        Z-Function: member[1] = negative

centerVertex = -0.611
                                        rightVertex = 0.047

        Tiangular: member[2] = zero

                                        leftVertex   = -0.297
                                        centerVertex = -0.178
                                        rightVertex = 0.471

        S-Function: member[3] = positive

                                        leftVertex   = 0.102
                                        centerVertex = 0.369

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

RUN 1 of 1: GENERATION 199->200
_____

Generation 199 Accumulated Statistics:
Total Crossovers = 3000, Total Mutations = 20714
min = 0.444444   max = 0.992721   avg = 0.812836   sum = 40.641800
Global Best Individual so far, Generation 191:
Fitness = 0.992738:
_____

Appendix E

Rule Space for Pendulum Balancing Controller

IF error IS negative THEN current IS negative END
IF error IS negative THEN current IS zero END
IF error IS negative THEN current IS positive END
IF error IS negative AND Derror IS negative THEN current IS negative END
IF error IS negative AND Derror IS negative THEN current IS zero END
IF error IS negative AND Derror IS negative THEN current IS positive END
IF error IS negative AND Derror IS zero THEN current IS negative END
IF error IS negative AND Derror IS zero THEN current IS zero END
IF error IS negative AND Derror IS zero THEN current IS positive END
IF error IS negative AND Derror IS positive THEN current IS negative END
IF error IS negative AND Derror IS positive THEN current IS zero END
IF error IS negative AND Derror IS positive THEN current IS positive END
IF error IS zero THEN current IS negative END
IF error IS zero THEN current IS zero END
IF error IS zero THEN current IS positive END
IF error IS zero AND Derror IS negative THEN current IS negative END
IF error IS zero AND Derror IS negative THEN current IS zero END
IF error IS zero AND Derror IS negative THEN current IS positive END
IF error IS zero AND Derror IS zero THEN current IS negative END
IF error IS zero AND Derror IS zero THEN current IS zero END
IF error IS zero AND Derror IS zero THEN current IS positive END
IF error IS zero AND Derror IS positive THEN current IS negative END
IF error IS zero AND Derror IS positive THEN current IS zero END
IF error IS zero AND Derror IS positive THEN current IS positive END
IF error IS positive THEN current IS negative END
IF error IS positive THEN current IS zero END
IF error IS positive THEN current IS positive END
IF error IS positive AND Derror IS negative THEN current IS negative END
IF error IS positive AND Derror IS negative THEN current IS zero END
IF error IS positive AND Derror IS negative THEN current IS positive END
IF error IS positive AND Derror IS zero THEN current IS negative END
IF error IS positive AND Derror IS zero THEN current IS zero END
IF error IS positive AND Derror IS zero THEN current IS positive END
IF error IS positive AND Derror IS positive THEN current IS negative END
IF error IS positive AND Derror IS positive THEN current IS zero END
IF error IS positive AND Derror IS positive THEN current IS positive END
IF Derror IS negative THEN current IS negative END
IF Derror IS negative THEN current IS zero END
IF Derror IS negative THEN current IS positive END
IF Derror IS zero THEN current IS negative END
IF Derror IS zero THEN current IS zero END
IF Derror IS zero THEN current IS positive END

IF Derror IS positive THEN current IS negative END
IF Derror IS positive THEN current IS zero END
IF Derror IS positive THEN current IS positive END

Appendix F

Rules Evolved by the Genetic Algorithm

IF error IS negative THEN current IS positive END

IF error IS zero AND Derror IS negative THEN current IS positive END

IF error IS zero AND Derror IS zero THEN current IS zero END

IF error IS positive THEN current IS negative END

IF error IS positive AND Derror IS negative THEN current IS positive END

IF Derror IS positive THEN current IS negative END

CONTROLLER: simplePendulum

--------------- INPUTS ---------------
number of input varibles = 2

variable[1] = 'error' and it has 3 members:

    Z-Function: member[1] = negative

                  centerVertex = -0.443
                  rightVertex = -0.172

    Tiangular: member[2] = zero

                  leftVertex = -0.955
                  centerVertex = 0.291
                  rightVertex = 0.301

    S-Function: member[3] = positive

                  leftVertex = -0.014
                  centerVertex = 0.938

lower bounds = -1.000000
upper bounds = 1.000000
delta = 2.000000
num. of bits = 10

variable[2] = Derror' and it has 3 members:

    Z-Function: member[1] = negative

                    centerVertex = -0.504
                    rightVertex = -0.172

    Tiangular: member[2] = zero

                    leftVertex   = -0.365
                    centerVertex = -0.244
                    rightVertex = 0.746

    S-Function: member[3] = positive

                    leftVertex   = -0.014
                    centerVertex = 0.680

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

--------------- OUTPUTS ---------------

number of output varibles = 1

variable[1] = 'current' and it has 3 members:

    Z-Function: member[1] = negative

                    centerVertex = -0.283
                    rightVertex = -0.152

    Tiangular: member[2] = zero

                    leftVertex   = -0.016
                    centerVertex = 0.102
                    rightVertex = 0.256

    S-Function: member[3] = positive

leftVertex   = -0.012
centerVertex = 0.566

lower bounds = -1.000000
upper bounds = 1.000000
delta   = 2.000000
num. of bits = 10

The number of criteria   = 36
Crite that were valid = 36

Mean Square Error = 0.039

Rule Space for Tank Recognition Problem

IF in1 IS low THEN class IS low END
IF in1 IS low THEN class IS med END
IF in1 IS low THEN class IS high END
IF in1 IS low AND in2 IS low THEN class IS low END
IF in1 IS low AND in2 IS low THEN class IS med END
IF in1 IS low AND in2 IS low THEN class IS high END
IF in1 IS low AND in2 IS low AND in3 IS low THEN class IS low END
IF in1 IS low AND in2 IS low AND in3 IS low THEN class IS med END
IF in1 IS low AND in2 IS low AND in3 IS low THEN class IS high END
IF in1 IS low AND in2 IS low AND in3 IS med THEN class IS low END
IF in1 IS low AND in2 IS low AND in3 IS med THEN class IS med END
IF in1 IS low AND in2 IS low AND in3 IS med THEN class IS high END
IF in1 IS low AND in2 IS low AND in3 IS high THEN class IS low END
IF in1 IS low AND in2 IS low AND in3 IS high THEN class IS med END
IF in1 IS low AND in2 IS low AND in3 IS high THEN class IS high END
IF in1 IS low AND in2 IS med THEN class IS low END
IF in1 IS low AND in2 IS med THEN class IS med END
IF in1 IS low AND in2 IS med THEN class IS high END
IF in1 IS low AND in2 IS med AND in3 IS low THEN class IS low END
IF in1 IS low AND in2 IS med AND in3 IS low THEN class IS med END
IF in1 IS low AND in2 IS med AND in3 IS low THEN class IS high END
IF in1 IS low AND in2 IS med AND in3 IS med THEN class IS low END
IF in1 IS low AND in2 IS med AND in3 IS med THEN class IS med END
IF in1 IS low AND in2 IS med AND in3 IS med THEN class IS high END
IF in1 IS low AND in2 IS med AND in3 IS high THEN class IS low END
IF in1 IS low AND in2 IS med AND in3 IS high THEN class IS med END
IF in1 IS low AND in2 IS med AND in3 IS high THEN class IS high END
IF in1 IS low AND in2 IS high THEN class IS low END
IF in1 IS low AND in2 IS high THEN class IS med END
IF in1 IS low AND in2 IS high THEN class IS high END
IF in1 IS low AND in2 IS high AND in3 IS low THEN class IS low END
IF in1 IS low AND in2 IS high AND in3 IS low THEN class IS med END
IF in1 IS low AND in2 IS high AND in3 IS low THEN class IS high END
IF in1 IS low AND in2 IS high AND in3 IS med THEN class IS low END
IF in1 IS low AND in2 IS high AND in3 IS med THEN class IS med END
IF in1 IS low AND in2 IS high AND in3 IS med THEN class IS high END
IF in1 IS low AND in2 IS high AND in3 IS high THEN class IS low END
IF in1 IS low AND in2 IS high AND in3 IS high THEN class IS med END
IF in1 IS low AND in2 IS high AND in3 IS high THEN class IS high END
IF in1 IS low AND in3 IS low THEN class IS low END
IF in1 IS low AND in3 IS low THEN class IS med END
IF in1 IS low AND in3 IS low THEN class IS high END

IF in1 IS low AND in3 IS med THEN class IS low END
IF in1 IS low AND in3 IS med THEN class IS med END
IF in1 IS low AND in3 IS med THEN class IS high END
IF in1 IS low AND in3 IS high THEN class IS low END
IF in1 IS low AND in3 IS high THEN class IS med END
IF in1 IS low AND in3 IS high THEN class IS high END
IF in1 IS med THEN class IS low END
IF in1 IS med THEN class IS med END
IF in1 IS med THEN class IS high END
IF in1 IS med AND in2 IS low THEN class IS low END
IF in1 IS med AND in2 IS low THEN class IS med END
IF in1 IS med AND in2 IS low THEN class IS high END
IF in1 IS med AND in2 IS low AND in3 IS low THEN class IS low END
IF in1 IS med AND in2 IS low AND in3 IS low THEN class IS med END
IF in1 IS med AND in2 IS low AND in3 IS low THEN class IS high END
IF in1 IS med AND in2 IS low AND in3 IS med THEN class IS low END
IF in1 IS med AND in2 IS low AND in3 IS med THEN class IS med END
IF in1 IS med AND in2 IS low AND in3 IS med THEN class IS high END
IF in1 IS med AND in2 IS low AND in3 IS high THEN class IS low END
IF in1 IS med AND in2 IS low AND in3 IS high THEN class IS med END
IF in1 IS med AND in2 IS low AND in3 IS high THEN class IS high END
IF in1 IS med AND in2 IS med THEN class IS low END
IF in1 IS med AND in2 IS med THEN class IS med END
IF in1 IS med AND in2 IS med THEN class IS high END
IF in1 IS med AND in2 IS med AND in3 IS low THEN class IS low END
IF in1 IS med AND in2 IS med AND in3 IS low THEN class IS med END
IF in1 IS med AND in2 IS med AND in3 IS low THEN class IS high END
IF in1 IS med AND in2 IS med AND in3 IS med THEN class IS low END
IF in1 IS med AND in2 IS med AND in3 IS med THEN class IS med END
IF in1 IS med AND in2 IS med AND in3 IS med THEN class IS high END
IF in1 IS med AND in2 IS med AND in3 IS high THEN class IS low END
IF in1 IS med AND in2 IS med AND in3 IS high THEN class IS med END
IF in1 IS med AND in2 IS med AND in3 IS high THEN class IS high END
IF in1 IS med AND in2 IS high THEN class IS low END
IF in1 IS med AND in2 IS high THEN class IS med END
IF in1 IS med AND in2 IS high THEN class IS high END
IF in1 IS med AND in2 IS high AND in3 IS low THEN class IS low END
IF in1 IS med AND in2 IS high AND in3 IS low THEN class IS med END
IF in1 IS med AND in2 IS high AND in3 IS low THEN class IS high END
IF in1 IS med AND in2 IS high AND in3 IS med THEN class IS low END
IF in1 IS med AND in2 IS high AND in3 IS med THEN class IS med END
IF in1 IS med AND in2 IS high AND in3 IS med THEN class IS high END
IF in1 IS med AND in2 IS high AND in3 IS high THEN class IS low END
IF in1 IS med AND in2 IS high AND in3 IS high THEN class IS med END
IF in1 IS med AND in2 IS high AND in3 IS high THEN class IS high END

IF in1 IS med AND in3 IS low THEN class IS low END
IF in1 IS med AND in3 IS low THEN class IS med END
IF in1 IS med AND in3 IS low THEN class IS high END
IF in1 IS med AND in3 IS med THEN class IS low END
IF in1 IS med AND in3 IS med THEN class IS med END
IF in1 IS med AND in3 IS med THEN class IS high END
IF in1 IS med AND in3 IS high THEN class IS low END
IF in1 IS med AND in3 IS high THEN class IS med END
IF in1 IS med AND in3 IS high THEN class IS high END
IF in1 IS high THEN class IS low END
IF in1 IS high THEN class IS med END
IF in1 IS high THEN class IS high END
IF in1 IS high AND in2 IS low THEN class IS low END
IF in1 IS high AND in2 IS low THEN class IS med END
IF in1 IS high AND in2 IS low THEN class IS high END
IF in1 IS high AND in2 IS low AND in3 IS low THEN class IS low END
IF in1 IS high AND in2 IS low AND in3 IS low THEN class IS med END
IF in1 IS high AND in2 IS low AND in3 IS low THEN class IS high END
IF in1 IS high AND in2 IS low AND in3 IS med THEN class IS low END
IF in1 IS high AND in2 IS low AND in3 IS med THEN class IS med END
IF in1 IS high AND in2 IS low AND in3 IS med THEN class IS high END
IF in1 IS high AND in2 IS low AND in3 IS high THEN class IS low END
IF in1 IS high AND in2 IS low AND in3 IS high THEN class IS med END
IF in1 IS high AND in2 IS low AND in3 IS high THEN class IS high END
IF in1 IS high AND in2 IS med THEN class IS low END
IF in1 IS high AND in2 IS med THEN class IS med END
IF in1 IS high AND in2 IS med THEN class IS high END
IF in1 IS high AND in2 IS med AND in3 IS low THEN class IS low END
IF in1 IS high AND in2 IS med AND in3 IS low THEN class IS med END
IF in1 IS high AND in2 IS med AND in3 IS low THEN class IS high END
IF in1 IS high AND in2 IS med AND in3 IS med THEN class IS low END
IF in1 IS high AND in2 IS med AND in3 IS med THEN class IS med END
IF in1 IS high AND in2 IS med AND in3 IS med THEN class IS high END
IF in1 IS high AND in2 IS med AND in3 IS high THEN class IS low END
IF in1 IS high AND in2 IS med AND in3 IS high THEN class IS med END
IF in1 IS high AND in2 IS med AND in3 IS high THEN class IS high END
IF in1 IS high AND in2 IS high THEN class IS low END
IF in1 IS high AND in2 IS high THEN class IS med END
IF in1 IS high AND in2 IS high THEN class IS high END
IF in1 IS high AND in2 IS high AND in3 IS low THEN class IS low END
IF in1 IS high AND in2 IS high AND in3 IS low THEN class IS med END
IF in1 IS high AND in2 IS high AND in3 IS low THEN class IS high END
IF in1 IS high AND in2 IS high AND in3 IS med THEN class IS low END
IF in1 IS high AND in2 IS high AND in3 IS med THEN class IS med END
IF in1 IS high AND in2 IS high AND in3 IS med THEN class IS high END

IF in1 IS high AND in2 IS high AND in3 IS high THEN class IS low END
IF in1 IS high AND in2 IS high AND in3 IS high THEN class ISmed END
IF in1 IS high AND in2 IS high AND in3 IS high THEN class IS high END
IF in1 IS high AND in3 IS low THEN class IS low END
IF in1 IS high AND in3 IS low THEN class ISmed END
IF in1 IS high AND in3 IS low THEN class IS high END
IF in1 IS high AND in3 ISmed THEN class IS low END
IF in1 IS high AND in3 ISmed THEN class IS med END
IF in1 IS high AND in3 ISmed THEN class IS high END
IF in1 IS high AND in3 IS high THEN class IS low END
IF in1 IS high AND in3 IS high THEN class ISmed END
IF in1 IS high AND in3 IS high THEN class IS high END
IF in2 IS low THEN class IS low END
IF in2 IS low THEN class IS med END
IF in2 IS low THEN class IS high END
IF in2 IS low AND in3 IS low THEN class IS low END
IF in2 IS low AND in3 IS low THEN class IS med END
IF in2 IS low AND in3 IS low THEN class IS high END
IF in2 IS low AND in3 IS med THEN class IS low END
IF in2 IS low AND in3 IS med THEN class IS med END
IF in2 IS low AND in3 IS med THEN class IS high END
IF in2 IS low AND in3 IS high THEN class IS low END
IF in2 IS low AND in3 IS high THEN class ISmed END
IF in2 IS low AND in3 IS high THEN class IS high END
IF in2 IS med THEN class IS low END
IF in2 IS med THEN class IS med END
IF in2 IS med THEN class IS high END
IF in2 IS med AND in3 IS low THEN class IS low END
IF in2 IS med AND in3 IS low THEN class IS med END
IF in2 IS med AND in3 IS low THEN class IS high END
IF in2 IS med AND in3 IS med THEN class IS low END
IF in2 IS med AND in3 IS med THEN class IS med END
IF in2 IS med AND in3 IS med THEN class IS high END
IF in2 IS med AND in3 IS high THEN class IS low END
IF in2 IS med AND in3 IS high THEN class ISmed END
IF in2 IS med AND in3 IS high THEN class IS high END
IF in2 IS high THEN class IS low END
IF in2 IS high THEN class ISmed END
IF in2 IS high THEN class IS high END
IF in2 IS high AND in3 IS low THEN class IS low END
IF in2 IS high AND in3 IS low THEN class ISmed END
IF in2 IS high AND in3 IS low THEN class IS high END
IF in2 IS high AND in3 ISmed THEN class IS low END
IF in2 IS high AND in3 ISmed THEN class IS med END
IF in2 IS high AND in3 ISmed THEN class IS high END

IF in2 IS high AND in3 IS high THEN class IS low END
IF in2 IS high AND in3 IS high THEN class IS med END
IF in2 IS high AND in3 IS high THEN class IS high END
IF in3 IS low THEN class IS low END
IF in3 IS low THEN class IS med END
IF in3 IS low THEN class IS high END
IF in3 IS med THEN class IS low END
IF in3 IS med THEN class IS med END
IF in3 IS med THEN class IS high END
IF in3 IS high THEN class IS low END
IF in3 IS high THEN class IS med END
IF in3 IS high THEN class IS high END

Appendix H

Rules Evolved by the Genetic Algorithm

IF in1 IS negative AND in3 IS negative THEN class IS positive END

IF in1 IS positive THEN class IS positive END

IF in2 IS negative AND in3 IS negative THEN class IS negative END

IF in2 IS negative AND in3 IS negative THEN class IS positive END

IF in2 IS positive AND in3 IS negative THEN class IS negative END

IF in3 IS negative THEN class IS negative END

IF in3 IS positive THEN class IS positive END

Mean Square Error = 0.414
Floor-Ceiling Mean Square Error = 0.498
Floor-Ceiling Percent Correctly Classified = 394/516 = 76.36

DISTRIBUTION LIST

AUL/LSE
Bldg 1405 - 600 Chennault Circle
Maxwell AFB, AL 36112-6424   1 cy

DTIC/OCP
8725 John J. Kingman Rd, Suite 0944
Ft Belvoir, VA 22060-6218   2 cys

AFSAA/SAI
1580 Air Force Pentagon
Washington, DC 20330-1580   1 cy

PL/SUL
Kirtland AFB, NM 87117-5776   2 cys

PL/HO
Kirtland AFB, NM 87117-5776   1 cy

Official Record Copy
PL/VTS/Ross Wainwright
Kirtland AFB, NM 87117-5776   2 cys

PL/VT
Dr. Janet Fender
Kirtland AFB, NM 87117-5776   1 cy